Nokia Service Router Linux
7215 Interconnect System
7220 Interconnect Router
7250 Interconnect Router
7730 Service Interconnect Router

Release 26.3

Software Installation Guide

3HE 22248 AAAA TQZZA
Edition: 01
March 2026

Nokia is committed to diversity and inclusion. We are continuously reviewing our customer documentation and consulting with standards bodies to ensure that terminology is inclusive and aligned with the industry. Our future customer documentation will be updated accordingly.

Table of contents

1 About this guide

This document describes how to install the Nokia Service Router Linux (SR Linux) in various environments. It defines the required prerequisites and procedures for how to install SR Linux software elements. Examples of commonly used commands are provided.

This document is intended for network technicians, administrators, operators, service providers, and others who need to understand how the software is installed and upgraded.

Note: This manual covers the current release and may also contain some content to be released in later maintenance loads. See the SR Linux Software Release Notes for information about features supported in each load.

Configuration and command outputs shown in this guide are examples only; actual displays may differ depending on supported functionality and user configuration.

1.1 Precautionary and information messages

The following are information symbols used in the documentation.

DANGER: Danger warns that the described activity or situation may result in serious personal injury or death. An electric shock hazard could exist. Before you begin work on this equipment, be aware of hazards involving electrical circuitry, be familiar with networking environments, and implement accident prevention procedures.

WARNING: Warning indicates that the described activity or situation may, or will, cause equipment damage, serious performance problems, or loss of data.

Caution: Caution indicates that the described activity or situation may reduce your component or system performance.

Note: Note provides additional operational information.

Tip: Tip provides suggestions for use or best practices.

1.2 Conventions

The Nokia SR Linux documentation uses the following command conventions:

- **Bold** type indicates a command that the user must enter.
- Input and output examples are displayed in `Courier` text.
- A vertical bar (|) indicates a mutually exclusive argument.
- Square brackets ([ ]) indicate optional elements.

- Braces ({ }) indicate a required choice. When braces are contained within square brackets, they indicate a required choice within an optional element.
- *Italic* type indicates a variable.

The following table outlines platform grouping conventions used in the SR Linux documentation suite.

Note: Some platforms in the 7250 IXR support mixed systems. For more information about mixed system support, see "Chassis types" in the *Configuration Basics Guide*.

Table 1: Platform grouping legend

Platform group	Description
7215 IXS	7215 IXS-A1
7220 IXR	All 7220 IXR platforms
7220 IXR-D*x*	7220 IXR-D1, 7220 IXR-D2, 7220 IXR-D2L, 7220 IXR-D3, 7220 IXR-D3L, 7220 IXR-D4, 7220 IXR-D5
7220 IXR-H*x*	7220 IXR-H2, 7220 IXR-H3, 7220 IXR-H4, 7220 IXR-H4-32D, 7220 IXR-H5-32D, 7220 IXR-H5-64D, 7220 IXR-H5-64O
7250 IXR[1]	7250 IXR platforms
7250 IXR Gen 2	7250 IXR-6, 7250 IXR-10
7250 IXR Gen 2c+	7250 IXR-6e with IMM2, 7250 IXR-10e with IMM2, 7250 IXR-X1b, 7250 IXR-X3b
7250 IXR Gen 3	7250 IXR-6e with IMM3, 7250 IXR-10e with IMM3, 7250 IXR-18e, 7250 IXR-X4
7250 IXR-6e/10e (mixed system)	7250 IXR-6e (mixed system)[2], 7250 IXR-10e (mixed system)[2]
7730 SXR	7730 SXR-1-32D, 7730 SXR-1d-32D, 7730 SXR-1x-44S

1 References to the 7250 IXR platform group may be appended with (including mixed systems) or (excluding mixed systems) to indicate mixed system support.

2 References to this platform as part of 7250 IXR (mixed system) indicate mixed system support of 7250 IXR Gen 2c+ (IMM2) and 7250 IXR Gen 3 (IMM3). That is, the 7250 IXR-6e and 7250 IXR-10e can hold and support both IMM2 and IMM3 at the same time.

2 What's new

This section lists the changes that were made in this release.

Table 2: What's new in Release 26.3.1

Topic	Location
DHCP option based ZTP selection	DHCP option based ZTP selection
Support for 7730 SXR-1-32D and 7250 IXR-X4 platforms	Hardware overview
Supports 7220 IXR-H4	Secure Boot
Supports 7220 IXR-H5	Measured Boot
SD card recovery using the tools command	SD card recovery using the tools command
Recovery procedure titles updated to indicate the boot method	• Performing SD card recovery for SD card-boot platforms • Performing SD card recovery for SSD-boot platforms • Performing USB recovery for SSD-boot platforms
ZTP using USB/SD in a non-secure environment	ZTP using USB/SD in a non-secure environment

3 SR Linux software overview

This chapter describes the basic software operations on the SR Linux.

3.1 File system layout

The file system is distributed and laid out as a closed-source third-party application on the OS. The following table lists the file system layout.

Table 3: File system layout

Path	Description
`/opt/srlinux/appmgr/*`	YAML configuration for Nokia-provided applications
`/opt/srlinux/appmgr/aaa_mgr`	System files for AAA manager
`/opt/srlinux/appmgr/logmgr`	System files for log manager
`/opt/srlinux/bin/*`	Application binaries
`/opt/srlinux/firmware/*`	Contains firmware (BIOS, IOCTL, CPLD, and so on)
`/mnt/nokiaos/<version>/imminit.tar`	IMM image
`/opt/srlinux/lib/*`	Nokia-provided shared libraries
`/opt/srlinux/models/*`	Nokia-provided YANG models
`/opt/srlinux/osync/*`	Configuration files and exclude/include files for overlay synchronization
`/opt/srlinux/python/*`	The virtual environment used to run SR Linux Python processes
`/opt/srlinux/systemd/*`	Systemd unit files and configuration
`/opt/srlinux/usr/*`	Community-provided binaries, shared libraries, and licenses
`/opt/srlinux/var/run/*`	Running directory for process sockets
`/opt/srlinux/ztp/*`	ZTP virtual environment, templates, and client
`/etc/opt/srlinux/config.json`	System configuration

Path	Description
`/etc/opt/srlinux/config.json.gz`	Compressed system configuration (if the configuration needs to compressed)
`/etc/opt/srlinux/banner`	The system banner, pre-login
`/etc/opt/srlinux/license.key`	License file
`/etc/opt/srlinux/srlinux.rc`	Global environment file
`/etc/opt/srlinux/tls/*`	User-configured certificates
`/etc/opt/srlinux/appmgr/*`	YAML configuration for operator-provided agents
`/etc/opt/srlinux/appmgr/overrides`	YAML overrides for operator and Nokia-provided applications
`/etc/opt/srlinux/models/*`	YANG modules for operator-provided agents
`/etc/opt/srlinux/checkpoint/*`	Configuration checkpoints
`/etc/opt/srlinux/devices/*`	Softlink to the devices directory: `/var/run/srlinux/devices/*`
`/etc/opt/srlinux/cli/plugins/*`	Operator-provided plug-ins
`/var/opt/srlinux/run/*`	Application PIDs
`/var/log/srlinux/buffer/*`	Tmpfs logging
`/var/log/srlinux/buffer.persist/*`	Persistent buffered logging
`/var/log/srlinux/file/*`	Persistent logging
`/var/log/srlinux/debug/*`	Debug logging, tmpfs
`/var/log/srlinux/debug.persist/*`	Persistent debug logging
`/var/log/srlinux/monitor/*`	Log_mgr tmpfs storage
`/var/log/srlinux/monitor.persist/*`	Log_mgr persistent storage
`/var/log/srlinux/archive/*`	Archive directory for previous startups

Path	Description
`/var/run/srlinux/devices/*`	Discovered devices
`$HOME/.srlinuxrc`	Per-user environment
`$HOME/srlinux/cli/plugins/*`	Per-user CLI plug-ins

The Solid State Drive (SSD) is used for an overlay file system, allowing the user to add persistent modifications to the system.

3.2 Boot process from an internal storage device (SD/SSD)

About this task

Note: This boot process applies only to the following platforms:

- 7250 IXR
- 7220 IXR
- 7730 SXR

The SR Linux boots using a normal Linux boot mechanism. The BIOS is set up to boot from the internal storage device. The following is the normal boot sequence:

Procedure

Step 1. The system powers on. Assuming a fully populated system, all components initialize to the point where they bring up their link on the back door bus. Fans are under hardware control and run at 100% speed.

Step 2. Both control modules start their boot sequence. During this sequence, the following occurs:

a. The BIOS tries to boot off the internal storage device.

b. Grub2 loads the kernel and initramfs into memory. The kernel command line points to squashfs of the root file system used to run SR Linux, including base Debian and SR Linux applications. The squashfs is unpacked and loaded.

c. When the squashfs root filesystem is loaded, the application manager (sr_app_mgr) starts and loads the applications based on their start order.

d. The system is operational.

Step 3. On control-redundant platforms, both control modules attempt to boot at the same time. The control module in slot 'B' waits up to 30 seconds before becoming active, after detecting slot 'A' on the back door bus.

Step 4. On multi-slot platforms, the sr_chassis_mgr and sr_device_mgr initialize, push images, and boot each line card and Switch Fabric Module (SFM). This includes making decisions based on power availability and taking control of fans.

3.3 Boot process from an embedded Multi-Media Card (eMMC)

About this task

Note: This boot process applies only to the 7215 IXS system.

The SR Linux boots using a normal Linux boot mechanism. The U-Boot is set up to boot from an eMMC. The following is the normal boot sequence. For starting SR Linux with factory defaults, see Starting SR Linux with factory defaults or post-ONIE installation.

Procedure

Step 1. The system powers on. The U-Boot loads the SR Linux Flattened Image Tree (FIT) image from an eMMC storage into memory, which contains the kernel, device tree, and initramfs. The kernel command line points to squashfs of the root file system used to run SR Linux, including base Debian and SR Linux applications. The squashfs is unpacked and loaded.

Step 2. After the squashfs root filesystem has loaded, the application manager (sr_app_mgr) starts and loads the applications based on their start order.

Step 3. The sr_device_mgr initializes the PSUs, fans, and LED system devices, upgrades firmware if required, and then the sr_app_mgr sequentially launches the remaining applications.

Step 4. The system is operational.

3.4 Secure Boot

SR Linux Secure Boot ensures that the software executed by the system is trusted and originated from Nokia IP routing.

At every boot of the control card, each step in the boot process verifies the digital signature of the next software element to boot for integrity and authenticity up to the SR Linux application contained in the squashfs root file system. This boot sequence forms the chain of trust for Secure Boot.

The Secure Boot chain is rooted in the platform control card firmware based on UEFI (Unified Extensible Firmware Interface) specifications. As such, Nokia Platform Key, Key Exchange Key, allowed and disallowed databases are provisioned in the system when Secure Boot is activated to perform the required signature verification.

Firmware updates are also digitally signed and verified using the same principle. The signature verification of a firmware update is performed at update application time by the existing firmware before the firmware update can proceed.

Software image signatures use RSA-4096 keys and SHA-384 hashes, and some signatures are based on SHA-256 hashes.

Supported platforms

Secure Boot is supported on the following platforms:

- 7220 IXR-D2L
- 7220 IXR-D3L

- 7220 IXR-H4
- 7220 IXR-H4-32D
- 7220 IXR-H5
- 7250 IXR-6e CPM4 with Root of Trust
- 7250 IXR-10e CPM4 with Root of Trust
- 7250 IXR-X1b
- 7250 IXR-X3b
- 7250 IXR-18e
- 7730 SXR

3.4.1 Activate Secure Boot

Procedure

Depending on the system, Secure Boot can be enabled from manufacturing or field enabled on compatible system control cards using CLI commands.

To activate Secure Boot on compatible control cards that do not have Secure Boot enabled from manufacturing, the following command is used providing the card slot, card serial number, and confirmation code:

```
tools platform trust secure-boot control <A|B> activate confirmation-code <string> serial-
number <string>
```

The card serial number and Secure Boot confirmation code are required to avoid accidental activation of Secure Boot in the network. The confirmation code is `secure-boot-permanent`.

The Secure Boot activate command verifies that the boot chain used at the next reboot of the system is properly signed otherwise the command returns an error in CLI.

WARNING:
After Secure Boot is activated on a control card, the capability is permanently enabled and cannot be disabled. The control card permanently refuses to execute unsigned software for security reasons. As a result, it is not possible to downgrade to a software release published before the release that introduced Secure Boot for a specific platform.

Example

The following example activates Secure Boot on slot A.

```
A:srl1# tools platform trust secure-boot control A activate confirmation-code secure-boot-
permanent serial-number NS22222222
```

The following example shows the warning messages and prompt returned when proceeding with Secure Boot activation:

```
WARNING: This operation will permanently activate secure boot on the card and cannot be
 reversed. The card will also be immediately rebooted. After activation, the system will
 only accept digitally signed software and will not boot using un-signed software.
Are you sure you want to activate secure boot and reboot this control module (y/[n])?y
```

3.4.2 Secure Boot state

Procedure

Secure Boot status is available per control card and is obtained using the following command:

show platform trust secure-boot control <slot> detail

Example: Check Secure Boot status

The following example checks the Secure Boot on slot A.

```
A:srl1# show platform trust secure-boot control A
  +------+--------------------+
  | Slot | Operational Status |
  +======+====================+
  | A    | enabled            |
  +------+--------------------+
```

The `Operational Status` indicates whether the Secure Boot is enabled or disabled.

Example: Check detailed Secure Boot state information command per control card

The **show platform trust secure-boot control <A|B> detail** command provides detailed Secure Boot state information based on the currently used Secure Boot UEFI variables and the modification dataset such as:

- `up-to-date status`: Status of the Secure Boot variables programmed in the control module compared to the current modification dataset
- `update-required status`: Indicates which variable require updating

```
A:Dut-A# show platform trust secure-boot control A detail
================================================================================================
  Show report for Secure Boot on Controller A
================================================================================================
  Summary
    Operational Status: enabled
  Database Variable Modification Dataset Status
    Modification Dataset Present     : true
    Modification Dataset Valid       : true
    Variables Up To Date             : true
    dbx Update Required              : false
    db Update Required               : false
    PK Update Required               : false
    KEK Update Required              : false
    Modification Dataset db Conflict : false
    Modification Dataset dbx Conflict: false
    Modification Dataset Digest      :
 0b5e6d97c0915ad9698634b2889da9fc37b3271d0f7914304c58f819ba037f6c
================================================================================================
  UEFI Security Database Variables
```

3.4.3 Update Secure Boot variables

In case the Secure Boot UEFI variables are updated, specific secure boot commands are used to update the allowed (`db`) and disallowed (`dbx`) databases programmed in the CPM firmware.

To activate or update the Secure Boot, the SR Linux software image contains a modification dataset that includes the UEFI variables, which can be installed on the control cards.

If a change to the UEFI `db` or `dbx` is included in a future software release, the operator must install the new UEFI variable modification dataset before deploying the new image. The UEFI variable modification dataset is included in the binary image of the new software, in the current SR Linux file system.

Use the following command to extract and install a secure boot variable update from the specified image file:

```
tools system secure-boot install-update <file> namespace <value>
```

where,

- `file`: refers to the file path or HTTP/HTTPS/SFTP URL for the image file. Default HTTP/HTTPS URL download mode is insecure.
- `namespace`: refers to the network namespace to use when downloading the image.

For example,

```
tools system secure-boot install-update srlinux.bin
```

Use the following command to display the installed update:

```
tools system secure-boot display-update file <value>
```

where, `file` refers to the file path of the secure boot variable update package. For example,

```
tools system secure-boot display-update file /etc/opt/srlinux/secure_boot_var_update.json
```

After installing the UEFI variable modification dataset, the operator must update the UEFI databases (`db`/`dbx`) programmed in the CPM firmware.

- To update the UEFI `db` per control card, use the following command:

  ```
  tools platform trust secure-boot control <A|B> update confirmation-code <string> serial-
  number <string>
  ```

- If a change to the UEFI `dbx` is included in the currently running software release, the operator must execute the following command per control card:

  ```
  tools platform trust secure-boot control <A|B> revoke confirmation-code <string> serial-
  number <string>
  ```

3.5 TPM Keys (IDevID and IAK)

The control cards (CPMs) are provisioned by Nokia at manufacturing time with identity keys and their associated x509 certificates signed by the Nokia Factory Certificate Authority (CA).

These keys and certificates are permanently programmed in the Trust Platform Module (TPM) of the control card and can be used for device authentication or attestation. The asymmetric cryptographic algorithm for the identity key depends on the platform: RSA-2048 for 7250 IXR CPM4 with RoT, 7250 IXR-X1b/X3b, and ECC P-384 for 7250 IXR-X4, 7730 SXR, 7250 IXR-18e, and 7220 IXR-H5 systems.

Supported Platforms

TPM keys are supported on the following platforms:

- 7220 IXR-H5
- 7250 IXR-6e CPM4 with Root of Trust
- 7250 IXR-10e CPM4 with Root of Trust
- 7250 IXR-X1b
- 7250 IXR-X3b
- 7250 IXR-18e
- 7250 IXR-X4
- 7730 SXR

Initial Device Identity (IDevID)

The factory-provisioned Initial Device Identity (IDevID) is a private key, and its associated certificate is signed by Nokia IP Routing Factory CA. This identity key and certificate are then used in software features to provide cryptographic evidence that the system used by the customer was manufactured by Nokia. The provisioning of the keys and certificates follows the Trusted Computing Group (TCG) TPM 2.0 Keys for Device Identity and Attestation specifications. The IDevID certificate binds the TPM to its control card by including the serial number of the control card in the certificate subject field. The IDevID can be used in secure remote bootstrap to authenticate the SR Linux router as a client and in the TLS server profile to authenticate the router acting as a server. On a system with a redundant control card, the active control card automatically verifies the standby card IDevID(s), certificate(s), and proof of ownership of their associated private keys, before permitting the standby card to performing synchronization with the active card.

Initial Attestation Key (IAK)

The factory-provisioned Initial Attestation Key (IAK) is a private key, and its associated certificate is signed by Nokia IP Routing Factory CA. Similar to IDevID, the IAK certificate contains the control card serial number, binding the TPM to the control card. The provisioning of the keys and certificates follows the Trusted Computing Group (TCG) TPM 2.0 Keys for Device Identity and Attestation specifications. The IAK is a TPM restricted signing key used to sign TPM quotes, thereby providing cryptographic evidence that the information produced by the TPM originated from a genuine discrete TPM and the expected system.

3.5.1 Viewing TPM certificates

Procedure

Each CPM is factory-provisioned with IDevID and IAK keys . The Endorsement Key (EK) is provisioned by the TPM manufacturer. The signing and provisioning of the IDevID/IAK certificate binds the TPM to a CPM unique identifier, the serial number and platform type.

Example: Viewing an IAK certificate on TPM

The command below shows an example of an IAK certificate that has been installed on the TPM.

```
# show platform trust tpm control B certificate initial-attestation-key
  Certificate initial-attestation-key :
     NV Index : 0x1c91000
      Subject : CN=3HE17011AB-NS224662905-IAK,O=Nokia,serialNumber=NS224662905
```

```
       Issuer : CN=Nokia PoC RSA Issuing,O=Nokia,street=600 Mountain Ave\, Ste 700,postal
Code=07974,L=Murray Hill,ST=NJ,C=US
  Fingerprint : 60:33:76:88:D6:E5:C3:30:8D:C1:47:6E:4C:F6:70:28:0F:C2:73:D5:C0:82:98
:2A:1C:AE:95:6F:B8:10:CF:C6
```

3.6 Measured Boot

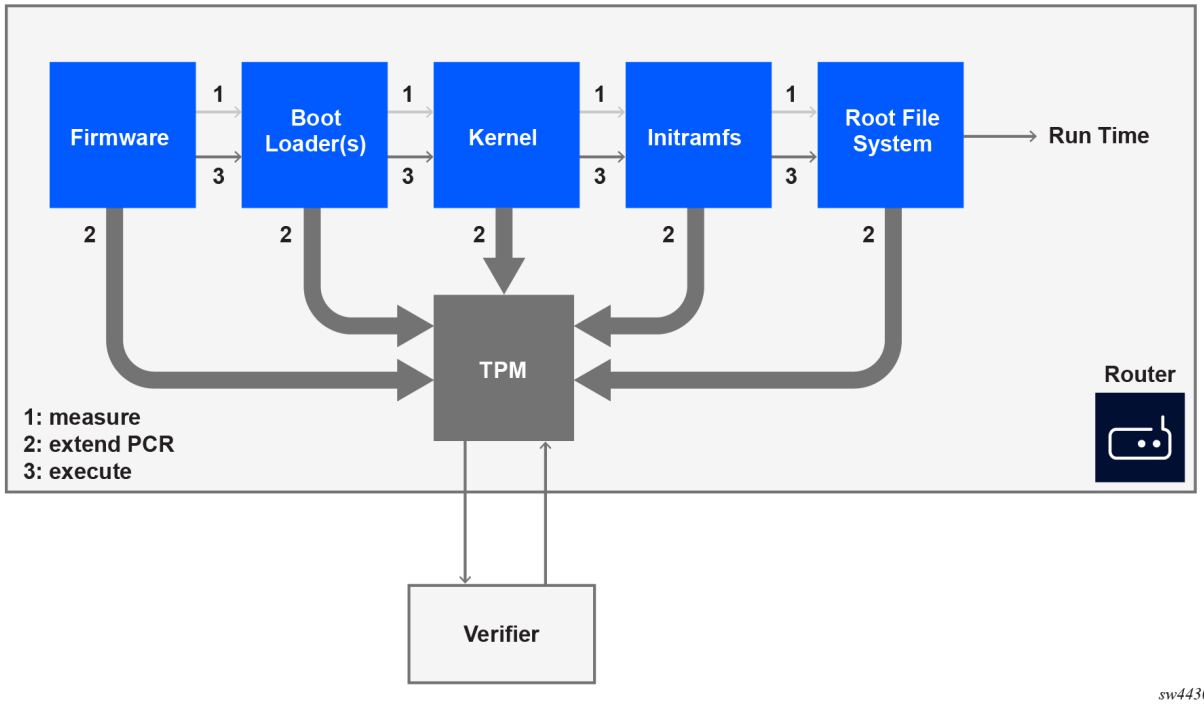
SR Linux control cards (CPMs) TPM 2.0 is the hardware root of trust for measurement and attestation in the context of Measure Boot. Measured Boot complements Secure Boot by providing cryptographic evidence that each piece of the system boot chain uses the software version allowed by the operator.

In a Measured Boot chain, each step in the boot process measures the next element before executing it. The initial boot firmware makes the first measurement in this boot chain and is the root of trust for measurements. Each measurement is a cryptographic hash value, and it is stored (extended) in the TPM Platform Configuration Register (PCR).

A measurement is extended in the TPM PCR instead of being directly written into it. The TPM performs this function by concatenating the previous hash value of the PCR with the new measurement. As a result, it is not possible to undo a measurement by design.

The boot process is described in the diagram below:

Figure 1: Boot process



After the router has established network connectivity, the operator can verify the integrity of the boot measurements as described in the remote attestation section.

Supported platforms

Measured Boot is supported on the following platforms:

- 7220 IXR-H5
- 7250 IXR-6e CPM4 with Root of Trust
- 7250 IXR-10e CPM4 with Root of Trust
- 7250 IXR-X1b
- 7250 IXR-X3b
- 7250 IXR-18e
- 7250 IXR-X4
- 7730 SXR

3.6.1 Viewing TPM PCR allocation

Procedure

SR Linux systems that contain TPM 2.0 compliant TPM devices use one or more of the `tpm20-hash-algo` cryptographic algorithms to hash the PCR measurements. An allocation refers to a list of banks and selected PCRs. This allocation is static for a given system.

Example: View TPM PCR allocation

In the following example, the TPM has two PCR banks, with both the SHA1 bank and SHA256 bank containing PCRs 0-23.

```
--{ candidate shared default }--[  ]--
# show platform trust tpm control A pcr-bank
  +---------+-------------------------------------------------------------------------------------------+
  |Hash     |                                    PCR Allocation                                         |
  |Algorithm|                                                                                           |
  +=========+===========================================================================================+
  |sha1     | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23 |
  |sha256   | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23 |
  +---------+-------------------------------------------------------------------------------------------+
```

3.6.2 PCR measurements

The SR Linux boot process and PCR usage are compliant with the Trusting Computing Group specifications, which include TPM/PCR initialization and Event Logs.

The boot phase events up to kernel run-time are recorded in the attestation BIOS log, as defined by the TCG.

At run-time, the Linux kernel Integrity Measurement Architecture (IMA) also records in PCR10 the measurements for anything executed or read by the root. The associated events are logged in the IMA log.

The PCRs relevant to software attestation are:

Table 4: Software attestation PCRs

PCR	Description
0	BIOS
2	UEFI driver/application Code
4	Boot Manager Code: SHIM, GRUB, kernel
6	Host Platform Manufacturer Specific
9	GRUB read/executed files: kernel, initramfs
14	Root File System: SquashFS

3.6.2.1 Displaying PCR values

Procedure

Operators can view TPM PCR measurements in the appropriate PCR registers using the command `tools platform trust attestation control <A|B> pcr-read pcr-selection <value>`, and compare them to the reference values, confirming the validity of the entire platform boot chain from firmware to SR Linux application launch. For information about reference values, see Reference values.

The following example retrieves the PCR measurement values from PCRs 0 to 10 from SHA256 bank.

```
# tools platform trust attestation control A pcr-read pcr-selection
 "sha256:0,1,2,3,4,5,6,7,8,9,10"
TPM PCR Read results for sha256:0,1,2,3,4,5,6,7,8,9,10
 sha256:
  0 : 0x43B4FD809823280E8A7D7A31C7D278CBD4A9994A13258A3D0E42C1940CC14222
  1 : 0x8E341EA8C2E1150C53613FB442340939B4BA54425309BFE4E9D71F7F44769753
  2 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
  3 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
  4 : 0x0D05C6F050C5CD50543438ACCE37868056D932DA0EF2D3624905CAE432E8231D
  5 : 0x8BEB965936843F26EDAC516D370F543AE1CB38506700160A25937AF1FCC8BE50
  6 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
  7 : 0x65CAF8DD1E0EA7A6347B635D2B379C93B9A1351EDC2AFC3ECDA700E534EB3068
  8 : 0xFAAB584987876D99A3D55E9ADDFFF6897C921F3759CE6DE020AC67E5EAAB351A
  9 : 0xC0885BD8B03BF30092AEACDE3ECCFCE88815EA2017B39C838BAD977BBD4F585A
  10 : 0x7D2F7D1077AD760E9CA75DB079CB5AEBDBA807378F120DF96FA531CFA56CBC6D
```

3.6.2.2 Retrieving BIOS and IMA event logs

Procedure

To ensure trust in each recorded event, an operator can use the event log to re-compute a specific PCR and compare it with the running system PCR value. This is useful for identifying which event is causing a discrepancy when the final PCR value is not as expected.

Example: Retrieving BIOS event logs

In the following example, BIOS event logs are retrieved from PCR 0.

```
# tools platform trust attestation control B log-retrieval bios display from 1 to 2
Event: 1
PCRIndex:      0
EventType:   0x8 (EV_S_CRTM_VERSION)
Digest Count - 2
Hash Algo - 4 (sha1)
Digest    - 1489f923c4dca729178b3e3233458550d8dddf29
Hash Algo - 11 (sha256)
Digest    - 96a296d224f285c67bee93c30f8a309157f0daa35dc5b87e410b78630a09cfc7
Event Data (base64): AAA=
Decode:
===============
Event: 2
PCRIndex:      0
EventType:   0x80000008 (EV_EFI_PLATFORM_FIRMWARE_BLOB)
Digest Count - 2
Hash Algo - 4 (sha1)
Digest    - 07459f28fdd346be4c9f179aae26998e82384625
Hash Algo - 11 (sha256)
Digest    - 6e460fa66a9abece4ea81a5267616fb00cd9ddf1b59a8d120a0b6695ec00c883
Event Data (base64): AADo/wAAAAAABgAAAAAA==
```

Example: Retrieving IMA event logs

Linux IMA does not have a known good final PCR value because the system constantly records new measurements into the IMA PCR.

In the following example, IMA event logs are retrieved from PCR 10.

```
# tools platform trust attestation control B log-retrieval ima display from 1 to 1
Event: 1
PCR: 10
Template Name: ima-ng
Digest type: sha256
Digest: 582f3a3b53db66797618d4d9de3891e8e34a1903cb262b545e20affe4fe8b432
Event Name: boot_aggregate
```

3.6.3 Reference values

Verifiers can assess the system health by comparing TPM PCR measurements to published reference values. The PCR reference value file (`known_good_pcr_values.json`) is bundled in the SR Linux software distribution for each release and is present in the `/mnt/nokiaos/<build binary.bin>/` directory.

Note: Reference values only include the final PCR value, not the individual PCR events value.

3.6.4 Remote Attestation / Integrity Verification

For remote attestation and integrity verification, a separate Verifier, operating in its secure environment, interrogates the network device to retrieve a quote of the PCRs relevant to the attestation operation, and optionally, the Boot event log.

A quote created by the TPM is a TCG (Trusted Computing Group) defined structure that contains a digest of the PCRs selected and signed by the Initial Attestation Key (IAK) to provide evidence that it originated from the expected Nokia system.

3.6.4.1 Retrieving TPM PCR quotes

Procedure

SR Linux follows the challenge-response interaction model. The Verifier includes the `nonce` and `pcr-selection` parameters in its TPM quote request. A `nonce` is a base64 operator generated number intended to guarantee freshness and is used as part of a replay-protection mechanism. The `pcr-selection` is a `tpm2-tools` compliant string, it indicates the bank selection and the list of PCRs.

Example

In this example, a PCR quote is generated by using `nonce` value `"AQIDBAU="` and the PCR measurement values from PCRs 0, 1, 4, and 9 of SHA256 bank.

```
# tools platform trust attestation control A pcr-quote pcr-selection sha256:0,1,4,9 nonce
 AQIDBAU=
Quote Message
/1RDR4AYACIAC4FZZKr0a3ASLDLbqwprA0/
JqMST9CyW6qzJLpdNBb5aAAUBAgMEBQAAAAAfnxrfAAAA0QAAAAABAEoAQESgGgMAAAABAAsDEwIAACAumDsE26q
FifhCyLSD8BzWITTQ9VDPqyOstckkzocIJQ==
Quote Message Signature
ABQACwEACwKaKaIwRS89vjAxR0jYs94XeNoFPHDUMsbtCZFyxr6DhezIQDyNsi3t5p/XEZyGK9PuFOTXh7Loii
NsgBbxpXOxnbnFoB/iUv2yKg1xMd0zfbCpQXKSbtpWIevCwXsNonV/QTR8MhlX9CDAXF8Ij1okbKylQwwQrdm
+4EACHM+sWTuO0DuT6ncKU0tAujtcQEDpeABTiSNhZE81Fg4tPNaj5zAE1cs6QX1fqWB6TNtEuCeUM8SH9w
QbOKWXkwZDZDXZ9ssQ+XurO35DT3uS9xg3DJqd8t99yyh6jTO7nx/euV7hDr+KbBNVKCX0gsRCiIRBjqdyxbVu+
+iK3wZBFg==
Quoted PCRs
AQAAAAsAAxMCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAQAAAAQAAAAgAHh9RiQh7oJcUXXmYgBIutUzjeCKHoRvei9wF9zz/UmZAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAgADl/pI5hMi+sdRKNNFEZZ7S5klA6EwiSkl2OB1BzhzoAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAgAA0FxvBQxc1QVDQ4rM43hoBW2TLaDvLTYkkFyuQy6CMdAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAgAJOLh47xcSkxPpxHMgiizHEfaaEv4uFAG1hJ0TdOr76zAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=
```

4 Deploying SR Linux container images

This chapter describes SR Linux container installation topics. The container installation topics include:

- SR Linux container prerequisites
  Ensure that prerequisites are met before launching a container.
- Launching an SR Linux container manually
  Launches a single SR Linux container using a manual procedure.

4.1 SR Linux container prerequisites

Ensure that prerequisites are met before installing an SR Linux container.

Minimum system requirements:

- 4 GB RAM
- 2 v CPU
- The host machine user must have sudo privileges

Minimum software requirements:

- Docker CE installed, minimum version 18.09:
  https://docs.docker.com/get-docker/
- SR Linux container image downloaded from the https://github.com/nokia/srlinux-container-image repository.

4.2 Launching an SR Linux container manually

About this task

This procedure manually launches a single container.

Procedure

Step 1. Download the SR Linux container image from the SR Linux container image repository.

```
$ docker pull ghcr.io/nokia/srlinux:X.Y.Z
```

where *X* = major, *Y* = minor, *Z* = patch

Step 2. Check that the Docker image was imported correctly:

```
$ docker images
```

Example

```
REPOSITORY TAG          IMAGE ID              CREATED         SIZE
ghcr.io/nokia/srlinux 23.7.1         ebf0aaf37625          6 weeks ago     3.58GB
```

Step 3. Launch an instance of the SR Linux container on the host using the options in the following command line. This command must be entered in a single line.

```
sudo docker run -t -d --rm --privileged \ -u 0:0 \ -v srl23-7-1.json:/etc/opt/srlinux/config.json \ --
name srlinux ghcr.io/nokia/srlinux:23.7.1 \ sudo bash /opt/srlinux/bin/sr_linux
```

Note: Copying a long command string from a PDF file introduces line breaks. As a workaround, copy the text string and place into Notepad++. Highlight the text and select CTRL+J. The result is a single line with no returns.

Step 4. Check that the docker container has been created with the name 'srlinux':

```
$ docker ps
```

Example

```
CONTAINER ID  IMAGE               COMMAND             CREATED     STATUS   PORTS
NAMES
9d5dbd03f7f8  srlinux:23.7.1     "/tini--fixuid-q.." 3 mins ago  Up3 mins.
srlinux
```

Step 5. Open an SSH session to the DUT using the following credentials:

- username: `admin`
- password: `NokiaSrl1!`

Example

```
# ssh admin@$(docker inspect srlinux --format {{.NetworkSettings.IPAddress}})
admin@172.17.0.4's password:
     ___          ___                                ___          ___          ___
    /  /\        /  /\                    ___        /__/\        /__/\        /__/|
   /  /:/_      /  /::\    ___            /  /\       \  \:\       \  \:\      |  |:|
  /  /:/ /\    /  /:/\:\  /__/\          /  /:/        \  \:\       \  \:\     |  |:|
 /  /:/ /::\  /  /:/~/:/  \  \:\        /__/::\    _____\__\:\  ___  \  \:\  __|__|:|
/__/:/ /:/\:\/__/:/ /:/__\  \:\        \__\/\:\__ /__/::::::::\/__/\  \__\:\/__/::::\____
\  \:\/:/~/:/\  \:\/:::::/\  \:\ /  /:/ \  \:\/\ \  \:\~~\~~\\  \:\ /  /:/  ~\~~\::::/
 \  \::/ /:/  \  \::/~~~~  \  \:\  /:/   \  \:\/:/\  \:\  ~~~ \  \:\  /:/     |~~|:|~~
  \__\/ /:/    \  \:\       \  \:\/:/     \__\::/  \  \:\      \  \:\/:/      |  |:|
    /__/:/      \  \:\       \  \::/      /__/:/    \  \:\      \  \::/       |  |:|
    \__\/        \__\/        \__\/       \__\/      \__\/       \__\/        |__|/
Hello admin,
Welcome to the srlinux CLI.
Type 'help' (and press <ENTER>) if you need any help using this.
--{ running }--[  ]--
```

Step 6. Verify the application versions running on the system:

```
# show system application *
```

Example

```
--{ running }--[  ]--
# show system application *
+------------------+------+---------+------------------------+--------------------------+
|        Name      | PID  |  State  |       Version          |       Last Change        |
+==================+======+=========+========================+==========================+
| aaa_mgr          | 1209 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.097Z |
| acl_mgr          | 1220 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.097Z |
| app_mgr          | 1157 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.310Z |
```

```
| arp_nd_mgr         | 1231 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.097Z |
| bfd_mgr            |      | running |                         |                          |
| bgp_mgr            | 1608 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:12.022Z |
| chassis_mgr        | 1242 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.097Z |
| dev_mgr            | 1178 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:09.109Z |
| dhcp_client_mgr    | 1253 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.097Z |
| dhcp_relay_mgr     |      | running |                         |                          |
| dhcp_server_mgr    |      | running |                         |                          |
| ethcfm_mgr         |      | running |                         |                          |
| event_mgr          |      | running |                         |                          |
| evpn_mgr           | 1264 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.097Z |
| fhs_mgr            |      | running |                         |                          |
| fib_mgr            | 1275 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.097Z |
| gnmi_server        | 2323 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:16.809Z |
| gribi_server       |      | running |                         |                          |
| idb_server         | 1198 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:09.308Z |
| igmp_mgr           |      | running |                         |                          |
| isis_mgr           |      | running |                         |                          |
| json_rpc           | 2325 | running |                         | 2023-09-29T12:23:16.762Z |
| l2_mac_learn_mgr   | 1286 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.097Z |
| l2_mac_mgr         | 1297 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.097Z |
| l2_static_mac_mgr  |      | running |                         |                          |
| label_mgr          |      | running |                         |                          |
| lag_mgr            | 1313 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.097Z |
| ldp_mgr            |      | running |                         |                          |
| license_mgr        | 1324 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.097Z |
| linux_mgr          | 1341 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.097Z |
| lldp_mgr           | 2319 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:16.745Z |
| log_mgr            | 1354 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.097Z |
| macsec_mgr         |      | running |                         |                          |
| mcid_mgr           | 1365 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.097Z |
| mfib_mgr           | 1376 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.098Z |
| mgmt_server        | 1387 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.098Z |
| mirror_mgr         |      | running |                         |                          |
| mpls_mgr           |      | running |                         |                          |
| mplsoam_mgr        | 1398 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.098Z |
| net_inst_mgr       | 1416 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.098Z |
| oam_mgr            |      | running |                         |                          |
| oc_mgmt_server     |      | running |                         |                          |
| ospf_mgr           |      | running |                         |                          |
| p4rt_server        |      | running |                         |                          |
| pim_mgr            |      | running |                         |                          |
| plcy_mgr           | 1617 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:12.012Z |
| qos_mgr            | 1626 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:12.036Z |
| radius_mgr         | 1434 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.098Z |
| sdk_mgr            | 1446 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.098Z |
| segrt_mgr          |      | running |                         |                          |
| sflow_sample_mgr   | 1457 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.098Z |
| snmp_server-mgmt   | 2405 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:18.809Z |
| static_route_mgr   |      | running |                         |                          |
| supportd           | 511  | running |                         | 2023-09-29T12:23:08.652Z |
| te_mgr             |      | running |                         |                          |
| tepolicy_mgr       |      | running |                         |                          |
| twamp_mgr          |      | running |                         |                          |
| vrrp_mgr           |      | running |                         |                          |
| vxlan_mgr          | 1635 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:12.071Z |
| xdp_lc_1           | 1472 | running | v23.7.1-163-gd408df6a0c | 2023-09-29T12:23:10.098Z |
+--------------------+------+---------+-------------------------+--------------------------+
```

5 Installing software

This chapter describes software installation tasks. Software installation topics include:

- Hardware overview
  Describes the chassis variants of each system type. Software installations can be performed on each chassis variant.
- Installation overview
  Describes concepts to be familiar with before installing or upgrading.
- Performing software upgrades
  Upgrades the SR Linux software on 7250 IXR, 7220 IXR, 7730 SXR, and 7215 IXS systems.
- Performing SD card recovery for SD card-boot platforms
  Installs the SR Linux software on 7250 IXR and 7730 SXR systems.
- Performing SD card recovery for SSD-boot platforms
  Installs the SR Linux software on 7250 IXR-X1b/X3b/X4 systems.
- Performing USB recovery for SSD-boot platforms
  Installs the SR Linux software on 7220 IXR-H4/H5/DL/H4-32D systems.
- Bootstrapping using ONIE (7220 IXR-H series)
  Installs the SR Linux software on all 7220 IXR-Dx/Hx platforms except 7220 IXR-H4/H5/DL/H4-32D platforms.
- Bootstrapping using ONIE (7215 IXS system)
  Installs the SR Linux software on 7215 IXS systems.

5.1 Hardware overview

SR Linux can be installed on the 7250 IXR, 7215 IXS, 7220 IXR, and 7730 SXR systems, each with multiple chassis variants. The following sections reference the system types collectively with software installations applicable to all chassis variants.

Note: In the list below, platforms are grouped (and those groups are named) to align with their respective hardware installation guides. Each router series has a dedicated installation guide containing complete specifications, recommendations for preparing the installation site, and procedures to install and ground the routers. You can find a list of the chassis installation guides in the “SR Linux documentation” section of the *SR Linux Product Overview Guide*.
This method of grouping platforms differs from the convention used elsewhere in the SR Linux product documentation, which is described in Table 1: Platform grouping legend.

The following systems are collectively referred to as 7250 IXR systems:

- 7250 IXR-6
- 7250 IXR-6e
- 7250 IXR-10
- 7250 IXR-10e

- 7250 IXR-X1b
- 7250 IXR-X3b
- 7250 IXR-18e
- 7250 IXR-X4

The following systems are collectively referred to as 7220 IXR-D systems:

- 7220 IXR-D1
- 7220 IXR-D2
- 7220 IXR-D3
- 7220 IXR-D4
- 7220 IXR-D5

The following systems are collectively referred to as 7220 IXR-DL systems:

- 7220 IXR-D2L
- 7220 IXR-D3L

The following systems are collectively referred to as 7220 IXR-H*x* systems:

- 7220 IXR-H2
- 7220 IXR-H3
- 7220 IXR-H4
- 7220 IXR-H4-32D
- 7220 IXR-H5-32D
- 7220 IXR-H5-64D
- 7220 IXR-H5-64O

The 7220 IXR series includes the 7220 IXR-D, 7220 IXR-DL, and 7220 IXR-H*x* platforms.

The 7215 IXS series includes the 7215 IXS-A1 platform.

The following systems are collectively referred to as 7730 SXR systems:

- 7730 SXR-1d-32D
- 7730 SXR-1x-44S
- 7730 SXR-1-32D

For information about each chassis, see the *SR Linux Product Overview Guide*.

5.2 Installation overview

SR Linux can be installed on the 7250 IXR, 7220 IXR, 7215 IXS, and 7730 SXR systems.

Installations can be completed using the CLI. To perform either an initial imaging, reinstallation, or an upgrade or downgrade of a system, the operation requires pushing the new image to the device, changing the boot configuration, and rebooting.

Note: Nokia recommends performing a software upgrade or downgrade on the system console device, where there is physical access, as remote connectivity may be unavailable if an

issue occurs during the process. Note that configuration translation is not supported during downgrades.

In the installation procedure examples, commands preceded by **$** require root privilege. Commands preceded by **#** should be executed from a Bash shell.

The basic installation actions performed on the system do not change, regardless of the method used to install the SR Linux (either using the CLI or manually), but the CLI method is dependent on having a working system whereas the manual method is not.

5.2.1 Software image contents

The software image is a set of files provided as part of an SR Linux distribution. The files contained in an image are:

squashfs	Contains the SR Linux root file system, including any needed binaries for system operation.
initramfs (or initrd)	Contains an initial file system that is used to make the hardware operational before unpacking the SR Linux squashfs into memory, then switching the root file system to it.
kernel (or vmlinuz)	The Linux kernel is the initial program executed by the boot loader. The kernel handles all interactions between the OS and hardware.

To perform an installation, you must have an SR Linux image, which is a bin containing these files, along with several other files used for operations and maintenance.

5.2.2 Installation concepts

SR Linux boot locations vary by platform model.

- 7250 IXR-6/10/6e/10e/18e: Boots from an internal SD card
- 7730 SXR: Boots from the front panel SD card
- 7250 IXR-X1b/X3b/X4, 7220 IXR: Boots from an internal SSD. No other boot devices may be used with the system.
- 7215 IXS system: Boots from an embedded MultiMediaCard (eMMC)

The internal SD or SSD contains the following:

- a EFI system partition containing the EFI boot loader chain
- a partition used for SR Linux images
- two overlay partitions used for persistent storage

Installations can be performed manually without using the CLI. The process may also require partitioning an SD card external to the system, installing the EFI boot loader chain, and copying the SR Linux image to the device. Use of the manual method requires advanced knowledge of Linux commands, including disk formatting, copying files, unpacking compressed images, and editing of text files. Basic knowledge of editing text files in Linux is mandatory. The manual method requires a Linux server, with an empty SD card mounted (or use of a USB-SD card adapter).

For a 7215 IXS system, the U-Boot loads the SR Linux Flattened Image Tree (FIT) image from an eMMC storage into memory, which contains the kernel, device tree, and initramfs.

5.3 Performing software upgrades

This section describes methods to upgrade the software using the CLI. They require a working system, with SR Linux operational and the CLI available.

Note: Nokia recommends performing a software upgrade or downgrade on the system console device, where there is physical access, as remote connectivity may be unavailable if an issue occurs during the process. Note that configuration translation is not supported during downgrades.

Software upgrade options include:

- Software upgrade using the tools command
  Upgrades and deploys the software using the **tools system deploy-image** command. This method is supported on all 7250 IXR, 7220 IXR, 7730 SXR, and 7215 IXS systems.
- Software upgrade from the Bash shell
  Upgrades the software from the Bash shell using the CLI. This method is supported on all 7250 IXR, 7220 IXR, 7730 SXR, and 7215 IXS systems.
- In-service software upgrade
  Upgrades software across maintenance releases within the same major release (in conjunction with a warm reboot). This method is supported on 7220 IXR-D2, 7220 IXR-D3, 7220 IXR-D2L, and 7220 IXR-D3L systems only.

5.3.1 Software upgrade using the tools command

You can upgrade and deploy a new software image using the **tools system deploy-image** command in the CLI. With this command, there are two methods you can use to deploy an image. You can choose to deploy using an HTTP/HTTPS link to the software, or you can copy the image bin file onto the system, then deploy it.

Run the **tools system deploy-image** command with or without a **reboot** option. The **reboot** option deploys the image and reboots the system automatically to bring up the specified image. If the **reboot** option is not added, the image is only deployed. To then perform the upgrade, the system must be rebooted separately using the **tools platform chassis reboot** command.

The system identifies directories in `/mnt/nokiaos/` to remove based on the image versions. During installation, the system automatically deletes old unused image directories.

5.3.1.1 Software upgrade using a HTTP/HTTPS link

About this task

Deploy the image using an HTTP/HTTPS link to the software image.
The image download is insecure by default with cert verification disabled. Use the **tools system deploy-image** *<http link to bin file>* **reboot** command, where the *<http link to bin file>* links to the image.

Following the upgrade, the upgraded configuration is not saved automatically to be the startup configuration. See the "Configuration upgrades" section in the *SR Linux Configuration Basics Guide* for information about how to persist the upgraded configuration to disk.

Example: Upgrade using a HTTP/HTTPS link

In the following example, the **reboot** option is not used. After the image is deployed, the system must be rebooted separately.

```
# tools system deploy-image https://username:password@example.com/repository/srlinux-os/
21.3.0/srlinux-21.3.0-384.bin
Downloading with the srbase-mgmt namespace. Connection timeout: 5 seconds
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  782M    0  782M    0     0  99.9M      0 --:--:--  0:00:07 --:--:--  101M
Deploying SRL image version 21.3.0-384
2021:03:15 21:51:46:10 | EVENT | Version of new image 21.3.0-384
2021:03:15 21:51:46:10 | EVENT | Current version: 21.3.0-377, New version: 21.3.0-384
2021:03:15 21:52:21:10 | EVENT | Invoked sync call. It may take few seconds to complete.
2021:03:15 21:52:30:15 | EVENT | Syncing image with standby
Successfully deployed SRL image version 21.3.0-384
```

5.3.1.2 Software upgrade using the image bin file

About this task

Copy the image bin file onto the system, then use the **deploy-image** command after the bin file is uploaded.
Use the `tools system deploy-image <absolute path to bin file> reboot` command, where <*absolute path to bin file*> specifies the location of the bin.

Following the upgrade, the upgraded configuration is not saved automatically to be the startup configuration. See the "Configuration upgrades" section in the *SR Linux Configuration Basics Guide* for information about how to persist the upgraded configuration to disk.

Example: Upgrade using a bin file location

In the following example, the **reboot** option is used. After the image is deployed, the system reboots automatically to bring up the image.

```
# tools system deploy-image /tmp/srlinux-21.3.0-382.bin reboot
Deploying SRL image version 21.3.0-382
2021:03:16 21:08:17:57 | EVENT | Version of new image 21.3.0-382
2021:03:16 21:08:17:58 | EVENT | Current version: 21.3.0-388, New version: 21.3.0-382
2021:03:16 21:08:53:77 | EVENT | Invoked sync call. It may take few seconds to complete.
2021:03:16 21:09:01:73 | EVENT | Syncing image with standby
Successfully deployed SRL image version 21.3.0-382
--{ candidate shared default }--[  ]--
# 2021:03:16 21:10:27:17 | EVENT | Linux sync call executing
2021:03:16 21:10:27:21 | EVENT | Umount /dev/sdb1
2021:03:16 21:10:27:23 | EVENT | Umount /dev/sdb2
2021:03:16 21:10:27:26 | EVENT | sr_cli chassis reboot force requested.
21-03-16 14:10:28.472 sr_device_mgr: Chassis reboot force requested - rebooting
21-03-16 14:10:36.079 sr_device_mgr: Rebooting - chassis reboot requested
```

5.3.2 Software upgrade from the Bash shell

About this task

This procedure upgrades the software from the Bash shell using the CLI.

Procedure

Step 1. Copy the SR Linux image to a location that the system being installed has access to, either to a USB or SD card, or somewhere on the network (assuming that the system being installed has access to the server). Enter:

```
# cp <path-to-srlinux-image-bin> <destination-directory>
```

Example

```
# cp SRLinux-21.3.0-459.bin /mnt/removable
```

Step 2. Log in to the system being upgraded and start the Bash shell.

```
# ssh <user>@<address>
# bash
```

Example

```
# ssh linuxadmin@192.168.0.1
(linuxadmin@192.168.0.1) Password:
# bash
```

Step 3. Copy the image to the system. Do either of the following:

- If not using removable media (USB or SD card), copy the image to the system across the network:

  ```
  # sudo ns_exec srbase-mgmt bash
  # sudo scp <user>@<server-with-srlinux-image>:<path-to-srlinux-image-bin> <local-destination>
  ```

  Example:

  ```
  # sudo ns_exec srbase-mgmt bash
  # sudo scp serveruser@192.168.0.2:srlinux-21.3.0-459.bin /local-destination
  ```

- If using removable media (USB or SD card), insert either the USB or SD card into the system and mount it to a temporary directory:

  ```
  # sudo mkdir -p /mnt/removable
  # sudo mount <path-to-disk> /mnt/removable
  ```

  Example:

  ```
  # sudo mkdir -p /mnt/removable
  # sudo mount /dev/sdc1 /mnt/removable
  ```

Step 4. Unpack the SR Linux image to a location that the system being installed has access to, either across the network, or to a USB or SD card that may be inserted into the active control plane module:

```
# sudo mkdir -p /mnt/nokiaos/<version>
# sudo cp <local-destination>/<srlinux-image-file.bin> /tmp/<srlinux-image-file.bin>
# sudo chmod +x /<tmp>/<srlinux-image-file.bin>
# sudo /tmp/<srlinux-image-file.bin> --target /mnt/nokiaos/<version> --noexec
```

Example

```
# sudo mkdir -p /mnt/nokiaos/21.3.0-459
# sudo cp /mnt/removable/srlinux-21.3.0-459.bin /tmp/srlinux-21.3.0-459.bin
# sudo chmod +x /tmp/srlinux-21.3.0-459.bin
# sudo /tmp/srlinux-21.3.0-459.bin --target /mnt/nokiaos/21.3.0-459 --noexec
```

Step 5. Start an SR Linux CLI session, and retrieve the current version of the software. Multiple images can be shown.

```
# info from state system boot image
```

Example

```
# sr_cli
# info from state system boot image
    system {
         boot {
            image [
               21.3.0-449
               20.6.1-10
                  ]
            }
          }
```

Note: The **info from state system boot image** output only lists images present in the grub.cfg file. The **tools system boot available-images** output lists all of the images available in the system.

Step 6. Display the list of images in the `/mnt/nokiaos` directory.

```
# tools system boot available-images
```

Example

```
# sr_cli
# tools system boot available-images
['21.3.0-449*', 20.6.1-10', '21.3.0-459']
```

Note: The asterisk (*) character indicates the current running version.

Step 7. Update the boot image list by reordering the current version behind the new version:

```
# tools system boot image [ <version> <old-version> ]
```

Example

```
# tools system boot image [ 21.3.0-459 21.3.0-449 20.6.1-10 ]
```

```
Boot order is updated with ['21.3.0-459', '21.3.0-449', '20.6.1-10']
```

Step 8. Enter the following command to save the configuration as the startup configuration.

Example

```
save startup
```

Step 9. Reboot the chassis:

```
# tools platform chassis reboot
```

Step 10. Log in to the device via SSH or console, and confirm the new version.

Step 11. Following the upgrade, the upgraded configuration is not saved automatically to be the startup configuration. Enter the following command to save the configuration as the startup configuration.

Example

```
save startup
```

Note: See the "Configuration upgrades" section in the *SR Linux Configuration Basics Guide* for information about how to save new configuration upgrades.

Step 12. Nokia recommends that you remove the .bin file in the `/tmp` directory and old images in the `/mnt/nokiaos` directory manually after the system has successfully rebooted. The following example removes the `/tmp/srlinux-21.3.0-459.bin` file.

Example

```
# rm /tmp/srlinux-21.3.0-459.bin
```

5.3.3 In-service software upgrade

This section describes the In-Service Software Upgrade (ISSU) procedures you can use to upgrade the following systems:

- 7220 IXR-D2
- 7220 IXR-D3
- 7220 IXR-D2L
- 7220 IXR-D3L

Note: An ISSU cannot be used to perform a software downgrade.

Depending on the release version, you can perform either a minor ISSU or major ISSU. A minor ISSU updates software across maintenance releases within the same major release (for example, 25.10.1→25.10.2). A major ISSU updates software within the same major release year (for example, 27.3.1→27.7.1), and across consecutive major releases, as long as the source release is the Long-Term Release (LTR) of that year (for example, 25.10.1→26.10.1).

To perform an ISSU, the new target software image is identified, then the upgrade is performed in conjunction with a warm reboot to restart the system. During an ISSU upgrade, SR Linux maintains non-stop forwarding. A warm reboot brings down the control and management planes while the NOS reboots,

and graceful restart helpers assist with maintaining the forwarding state in peers. Any control plane or management plane functions are unavailable during a warm reboot, including the refreshing of neighbors, responding to ARP/ND, and any other slow path functions.

Warm reboot leverages control plane functionality to allow remote peers to continue forwarding based on the previously learned state. This process is known as graceful restart, where the remote system is the graceful restart helper, and SR Linux, when undergoing warm reboot, is being helped.

At a high level, the ISSU process requires the following steps. For a detailed ISSU procedure, see Performing an ISSU.

1. Back up the existing configuration.
2. Deploy the supported ISSU image using the **tools system deploy-image** command.
3. Update the boot images supported ISSU image with the **tools system boot image** command.
4. Ensure the running configuration is saved as the startup configuration with the **save startup** command.
5. Perform a **reboot warm** (with or without **force**).

5.3.3.1 Minor ISSU

Beginning in Release 21.6.1, you can perform a minor ISSU to update software across maintenance releases within the same major release. The upgrade does not require a datapath outage. For example, you can perform a minor ISSU in the following minor release versions. The upgrade can only be to a later version of the same minor release (when the later release becomes available).

Table 5: Minor ISSU

Minor release	Upgrade to	Examples
21.6	21.6.x	21.6.2, 21.6.3, and so on
21.11	21.11.x	21.11.2, 21.11.3, and so on
22.3	22.3.x	22.3.2, 22.3.3, and so on
22.6	22.6.x	22.6.2, 22.6.3, and so on
22.11	22.11.x	22.11.2, 22.11.3, and so on
23.3	23.3.x	23.3.2, 23.3.3, and so on
23.7	23.7.x	23.7.2, 23.7.3, and so on
23.10	23.10.x	23.10.2, 23.10.3, and so on
24.3	24.3.x	24.3.2, 24.3.3, and so on
24.7	24.7.x	24.7.2, 24.7.3, and so on
24.10	24.10.x	24.10.2, 24.10.3, and so on
25.3	25.3.x	25.3.2, 25.3.3, and so on
25.7	25.7.x	25.7.2, 25.7.3, and so on
25.10	25.10.x	25.10.2, 25.10.3, and so on

5.3.3.2 Major ISSU

Beginning in Release 25.10.1, you can perform major ISSUs to update software within the same major release year (for example, 27.3.1→27.7.1), and across consecutive major releases (for example, 25.10.1→26.10.1). Major ISSU upgrades require the source release to be the Long-Term Release (LTR) of that release year.

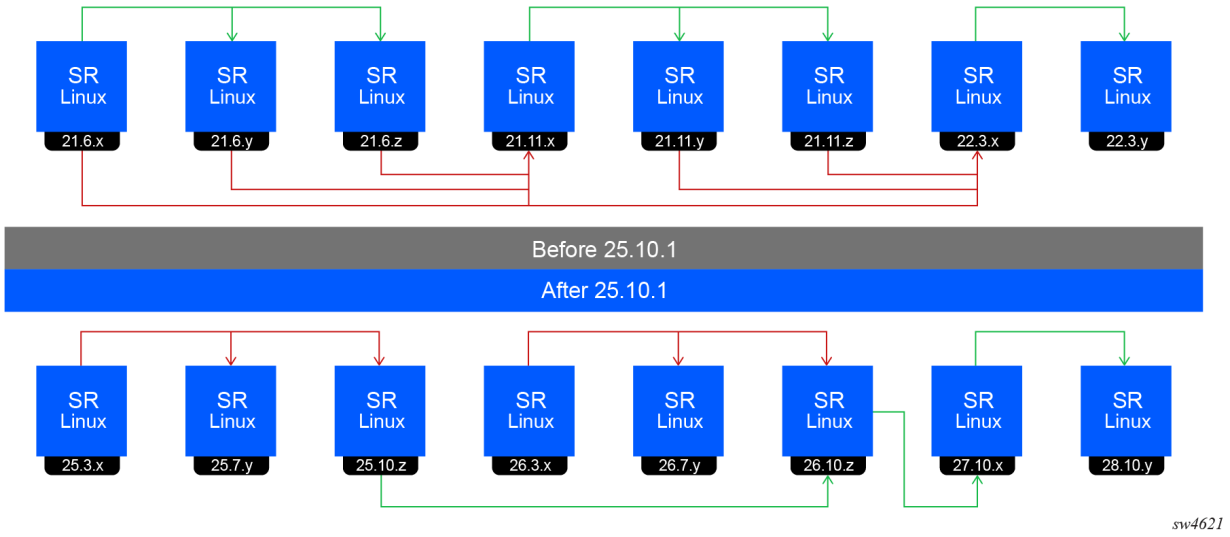
Note: As major ISSU was first introduced in 25.10.1, the following upgrade paths are not supported:

- 25.10.x→26.3.y
- 25.10.x→26.7.y
- 26.3.x→26.7.y
- 26.7.x→26.10.y

Table 6: Major ISSU

Major release	Upgrade to	Example
25.10.x	26.10.y	26.10.2, 26.10,3, and so on

The following diagram illustrates the required upgrade path for upgrading from 25.10.1 (the first supported release of the major ISSU) to 28.3.1. The green arrow indicates the supported upgrade paths and the red arrow indicates the non-supported upgrade paths.



1. **25.10.x → 26.10.y** – consecutive year major upgrade, moving from the LTR of one year to the LTR of the following year.
2. **26.10.y → 27.10.z** – consecutive year major upgrade, moving from the LTR of one year to the LTR of the following year.
3. **27.10.z → 28.3.1** – consecutive year major upgrade, moving from the LTR of the source year to the first major release of the following year.

5.3.3.3 Warm boot configuration support

Before performing a warm reboot as the final step of an ISSU, you must first confirm if the current SR Linux configuration supports warm reboot (including any destination image checks for ISSU). Use the **tools platform chassis reboot warm validate** command.

```
--{ running }--[  ]--
A:# tools platform chassis reboot warm validate
/platform:
    Warm reboot validate requested

/:
    Success
```

If the validation is successful, proceed with the warm reboot.

If a validation is unsuccessful, or if an attempt to perform a warm reboot fails, you can force the warm reboot using the additional **force** option. A warm reboot may not be successful if, for example, a peer does not support graceful restart. Force the warm reboot using the **tools platform chassis reboot warm force** command.

An unsuccessful validation or a failed warm reboot attempt cannot be forced using the additional **force** option in the following cases:

- The running configuration contains configuration paths that are not supported in ISSU. To complete the ISSU, invalid configuration paths must be removed from the running configuration. See YANG path support.
- The running configuration has not been saved as startup configuration.

Caution: Forcing a warm reboot may result in a service outage. The **force** option overrides any warnings, such as peers that are not configured, or peers that do not support graceful restart.

```
--{ running }--[  ]--
A:# tools platform chassis reboot warm force
/platform:
    Warm reboot force requested

/:
    Success
```

5.3.3.4 YANG path support

The following YANG paths must exist in your BGP configuration or via inheritance (if their context is present) for a warm reboot to succeed without an outage:

```
network-instance protocols bgp graceful-restart warm-reboot admin-state enable
network-instance protocols bgp group graceful-restart warm-reboot admin-state enable
network-instance protocols bgp neighbor graceful-restart warm-reboot admin-state enable
```

The following YANG paths must not exist in a configuration for a warm reboot to succeed without an outage:

```
interface hold-time
```

```
interface sflow
interface subinterface local-mirror-destination
interface subinterface ipv4 arp evpn
interface subinterface ipv6 neighbor-discovery evpn
interface subinterface type local-mirror-dest
network-instance protocols bgp evpn
network-instance protocols bgp group evpn
network-instance protocols bgp neighbor evpn
network-instance protocols isis
network-instance protocols ospf
network-instance vxlan-interface
system mirroring
system network-instance protocols evpn
system sflow
```

5.3.3.5 Performing an ISSU

About this task

You can perform an ISSU on 7220 IXR-D*x* systems only. Instead of rebooting the chassis to bring up the new software image, you will perform a warm reboot to conclude the upgrade. During the warm reboot, the system maintains non-stop forwarding.

The warm reboot process requires a minimum 100 MB of free disk space in the file system under `/etc/opt/srlinux`, excluding the files under the `warmboot/` directory. If the disk space is unavailable, the warm reboot will fail.

The examples in this section show an ISSU from SR Linux 21.6.1 to the next available maintenance release.

Note: When the control plane goes down during an ISSU, all SSH sessions are disconnected. Nokia recommends that you perform ISSU via a console session.

Note: Before you perform an ISSU, Nokia recommends you back up your existing configuration.

You can perform an ISSU upgrade in conjunction with the **tools system deploy-image** command. With this command, you can choose between two methods to deploy an image; you can choose to deploy using an HTTP/HTTPS link to the software, or you can copy the image bin file onto the system, then deploy it.

Procedure

Step 1. Using one of the methods described in Software upgrade using the tools command, deploy the new software image with the **deploy-image** command.

Step 2. Warm reboot the chassis to begin the upgrade. During the ISSU, the system maintains non-stop forwarding. The control plane goes down. Additionally, a **validate** option is also available to confirm that the current system configuration supports a warm boot.

```
# tools platform chassis reboot warm
```

Example

```
--{ running }--[  ]--
A:# tools platform chassis reboot warm
/platform:
    Warm reboot requested

/:
```

```
    Success

--{ running }--[  ]--
A:#
--{ [WARM BOOT] [FACTORY] running }--[  ]--
```

Step 3. The control plane comes back up and the SR Linux CLI is available again. Note the [WARM BOOT] indicator is still present in the banner as the upgrade is not yet complete.

Example

```
A:#
--{ [WARM BOOT] [FACTORY] running }--[  ]--
```

Step 4. When the warm reboot finishes, the ISSU is complete. The system will accept new configurations. The [WARM BOOT] indicator is no longer present in the banner.

Example

```
A:#
--{ running }--[  ]--
A:#
Current mode: running
```

Step 5. Use the **show version** command to confirm the new software image is running.

5.4 Starting SR Linux with factory defaults or post-ONIE installation

About this task

By default, the system boots with ZTP. To override the ZTP process and enable booting with the SR Linux image when the system is powered on, execute the following steps:

Procedure

Step 1. After the image is installed, the system reboots with the SR Linux image:

Example

7250 IXR-X1b/X3b reboot log.

```
[  OK  ] Finished shutdown_guard.service - Shutdown Guard.
[  OK  ] Finished sshd_key_remove.service - Remove old ssh key.
         Starting sr_boot.service - sr_boot...

SRLINUX 24.10.1-414
Kernel 6.1.25-28-amd64 on an x86_64

localhost login: 2024:08:25 17:36:13:78 | EVENT | Starting ZTP process
2024:08:25 17:36:13:80 | EVENT | Current version 24.10.1-414

localhost login:
localhost login:
localhost login:
localhost login: 2024:08:25 17:36:31:64 | EVENT | ZTP runtime remaining 3582.11
 seconds
2024:08:25 17:36:31:65 | EVENT | ZTP Perform DHCP_V4
```

```
localhost login:
```

Example

7220 IXR reboot log.

```
         Starting Wait for Plymouth Boot Screen to Quit...
         Starting Terminate Plymouth Boot Screen...
[  OK  ] Started Login Service.

SRLINUX 20.6.1-21398
Kernel 4.19.39-2.x86_64 on an x86_64

Localhost login: linuxadmin
Password: 2020:06:21 19:52:54:54 | EVENT | Starting ZTP process

[linuxadmin@localhost ~}$ system2020:06:21 19:52:58:64 | EVENT | Set link mgmt0 up
ctl disable z2020:06:21 19:53:03:82 | EVENT | ZTP Perform DHCP_V4. attempt[1]
t2020:06:21 19:53:04:14 | EVENT | Received dhcp lease on mgmt0 for 135.227.251.182/21
2020:06:21 19:53:04:23 | EVENT | option 66 provided by dhcp: http://135.277.248.118
2020:06:21 19:53:04:23 | EVENT | option 67 provided by dhcp: duts/SD-RD2-126/ztp-
config.yml
```

Example

7215 IXS reboot log.

```
         Starting kexec-load.servic…B: Load kernel image with kexec...
[  OK  ] Started sr_watchdog.service - SRLinux watchdog server.
[  OK  ] Started systemd-logind.service - User Login Management.
[  OK  ] Finished sr_shutdown.service - SRLinux shutdown action.
[  OK  ] Started kexec-load.service…LSB: Load kernel image with kexec.
[  OK  ] Finished ras-mc-ctl.servic….0.0 Drivers For Machine Hardware.
[  OK  ] Finished sshd_key_remove.service - Remove old ssh key.
         Starting sr_boot.service - sr_boot...
[  OK  ] Finished shutdown_guard.service - Shutdown Guard.

SRLINUX 24.10.1-248
Kernel 6.1.25-28-arm64 on an aarch64

localhost login: linuxadmin
Password:
Linux localhost 6.1.25-28-arm64 #1 SMP Mon Sep 16 18:06:18 UTC 2024 aarch64
[linuxadmin@localhost ~]$ 2024:10:18 01:22:07:15 | EVENT | Starting ZTP process
2024:10:18 01:22:07:18 | EVENT | Current version 24.10.1-248
```

Step 2. Enter the login credentials:

- username: `linuxadmin`
- password: `NokiaSrl1!`

Step 3. Disable the ZTP autoboot using the following command:

`ztp service stop --autoboot disable`

Example

```
$ ztp service stop --autoboot disable
Warning: This command to be used under guidance of Nokia Technical Support only.
 Unsupported usage can trigger instability of network
+--------+-----------------+
```

```
| key    | value           |
+--------+-----------------+
| status | Service stopped |
+--------+-----------------+
[linuxadmin@localhost ~]$
```

Step 4. After a reboot, the SR Linux service is started automatically.

Step 5. If the ZTP server is set up, the mgmt0 IP address is automatically configured by the DHCP server, as configuration for this is enabled by default.

Example

```
# info interface mgmt0
    interface mgmt0 {
        admin-state enable
        subinterface 0 {
            admin-state enable
            ipv4 {
                admin-state enable
                dhcp-client {
                }
            }
            ipv6 {
                admin-state enable
                dhcp-client {
                }
            }
        }
    }
```

Note: To manually configure the mgmt0 IP via CLI, you can add an IP address for it after deleting the `dhcp-client` (highlighted in bold) under the subinterfaces (`interface.subinterface.ipv4` and `interface.subinterface.ipv6`). You can add a default route using a static route. For more information, see the section "Configuring static routes" in *SR Linux Configuration Basics Guide.*

5.5 Performing SD card recovery for SD card-boot platforms

This section describes the recovery procedures for the 7250 IXR-6/10/6e/10e/18e, and 7730 SXR systems.

Note: For recovery procedures on 7250 IXR-X1b/X3b/X4 systems, see Performing SD card recovery for SSD-boot platforms.

5.5.1 Creating a bootable SD card

This section describes methods for creating a bootable SD card containing the SR Linux software image for use on a 7250 IXR-6/10/6e/10e/18e or 7730 SXR system. The supported methods are the following:

- Using a downloaded disk image
- Using another SD card
- Using a running SR Linux system

5.5.1.1 Using a downloaded disk image

About this task

You can create a bootable SD card containing the SR Linux software image for a 7250 IXR-6/10/6e/10e/18e or 7730 SXR system. This procedure requires a working Linux system running Debian/Ubuntu (Debian version 12 or higher, Ubuntu version 20 or higher) or CentOS (version 7 or higher).

Ensure you have access to a 16 GB SD card, which can be connected via a USB adapter if your server does not have an SD card slot. Before proceeding, format the SD card and remove any critical data, as the procedure erases all existing data.

In the following examples, `/dev/sdb` is used as the SD card device. All steps should be completed as a user with root privileges.

WARNING: If used incorrectly, this procedure could be destructive and may render the system creating the SD card inoperable. Verify the correct drive is being used before completing the installation.

Procedure

Step 1. Download the SR Linux image from the SR Linux software distribution package to the server being used to prepare the bootable SD card.

Step 2. Open a terminal and navigate to the directory where you downloaded the SR Linux image.

Step 3. Extract the image using the following command:

Example

```
gunzip -c srlinux-24.7.1.bin.sdcard-img.gz > srlinux-24.7.1.bin.sdcard-img
```

Step 4. Insert the SD card into the system to flash the SR Linux image.

Ensure the system correctly identifies the SD card because this action is destructive. The exact device path to your SD card depends on your Linux distribution and system setup. In this procedure, the SD card device is detected as `/dev/sdb`.

- If an SD card is connected through a USB adapter, the Linux system may identify it as `/dev/sdb` (with `/dev/sda` being a boot drive).
- If an SD card is connected to a device via an SD slot, the Linux system may identify it as `/dev/mmcblk0` (or `mmcblk1`, `mmcblk2`), depending on which MMC slot is used.

Note: If the system does not automatically detect the partition path, use the following commands to know the right partition path.

```
df -h
lsblk
```

If the correct path is unclear from the output, run the command both with and without the SD card inserted.

Step 5. Before flashing the SD card, unmount all partition paths. To find the partition path, run the command `df -h | grep /dev/sdb` and unmount all the partition paths displayed in the output using the following command:

```
# umount <dev-path-to-sdcard-partitions>
```

Example

```
# umount /dev/sdbX
```

Step 6. Flash the SD card with the SR Linux image file.

`sudo dd if=<machine>.srlinuxsdcard-img of=/dev/sdX bs=4M status=progress`
where *machine* = the image name for the device and *sdX* = the USB device name.

Example

The following is a command execution example on a Debian system.

```
sudo dd if=srlinux-24.7.1.bin.sdcard-img of=/dev/sdX bs=4M status=progress
```

Step 7. Umount the SD card. For instructions, see step 5.

Example

```
# umount /dev/sdbX
```

Step 8. Repeat steps 4 to 7 with another SD card for the standby control plane module (if applicable).

5.5.1.2 Using another SD card

About this task

Using a Linux machine running Debian/Ubuntu (version 12 or higher) or CentOS (version 7 or higher), you can copy an SR Linux image from one SD card to another SD card. You can then install the SR Linux software image on this second SD card onto a 7250 IXR-6/10/6e/10e/18e or 7730 SXR system.

Procedure

Step 1. Insert an SD card containing an SR Linux image into any Linux machine with a supporting SD slot. The SD card device is detected as `/dev/sdb` in this procedure example.

Ensure the system correctly identifies the SD card, because this action is destructive. The exact device path to your SD card depends on your Linux distribution and system setup.

- If an SD card is connected through a USB adapter, the Linux system identifies it as `/dev/sdb` (with `/dev/sda` being a boot drive).
- If an SD card is connected to a device via an SD slot, the Linux system identifies it as `/dev/mmcblk0` (or `mmcblk1`, `mmcblk2`), depending on which MMC slot is used.

Note: If the system does not automatically detect the partition path, use the following command to know the right partition path.

```
df -h
```

If the correct path is unclear from the output, run the command both with and without the SD card inserted.

Step 2. When the SD card is detected, copy the SR Linux image to the Linux machine:

```
sudo dd if=/dev/sdb of=sd.img status=progress
```

Step 3. Remove the SD card from the Linux machine.

Step 4. Insert the second SD card (to which the image is copied) into the Linux machine.

Step 5. Copy the SR Linux image from the Linux machine to the second SD card:

```
sudo dd if=sd.img of=/dev/sdb status=progress
```

Step 6. Remove the second SD card from the Linux machine.

Insert this SD card into the internal SD card slot of a 7250 IXR-6/10/6e/10e/18e or 7730 SXR system's control plane module. The system powers on with the image.

5.5.1.3 Using a running SR Linux system

About this task

You can create a bootable SD card directly on a 7250 IXR-6/10/6e/10e/18e or 7730 SXR system that is running SR Linux. This SD card can then be moved to and used in a different system.

Procedure

Step 1. Log in to the system running SR Linux via a console connection.

Step 2. Reboot the system and select `srlinux-rescue` from the image boot menu.

Step 3. Copy the `srlinux-xxx.bin` file using SCP.

To allow this, one of the ports on the system in the rescue image should have management connectivity. If there is no IP assigned to any of the ports automatically, you can add an IP manually using the **ifconfig** command:

`ifconfig <port> <ip address> netmask <ip address>`

Example

```
ifconfig eth4 192.168.255.254 netmask 255.255.255.0
```

Step 4. Find the target SD card device.

In this procedure example, the SD card device is detected as `/dev/sdb`.

Step 5. Find the current internal SD card device.

In this procedure example, the SD card device partition is detected as `/dev/sdc1`.

Step 6. Run the following command to flash the SD card device with the `srlinux-xxx.bin` file:

`bash <srlinux-xxx.bin> -- --dev <target SD device> --no-onie --source-efi <internal SD device>`

Example

```
bash srlinux-21.3.0-459.bin -- --dev /dev/sdb --no-onie --source-efi /dev/sdc1
```

Step 7. Remove the SD card from the system. Insert it into the internal SD card slot on the control plane module of the system where the software image is to be installed.

Step 8. Power on the system with the new image.

5.5.1.4 SD card recovery using the tools command

To create a recovery SD card on the SR Linux platform with an external slot, insert the SD card you want the image on into the external SD slot before running the **tools.system.restore-sd** command in the CLI.

This command can be run from a normally booted SR Linux image (the system-recovery image is not required).

Execute the **tools.system.restore-sd** command with the **current** option. The **current** option writes the active host image to the external SD card. The command automatically detects the removable SD card, confirms it is not the internal SD or SSD, and uses the running image as the source.

Note: The recovery SD card should only be used on the same platform type.

5.6 Performing SD card recovery for SSD-boot platforms

On 7250 IXR-X1b/X3b/X4, 7220 IXR-H4/H5/DL/H4-32D systems, Secure Boot is enabled by default, and the system boots from SSD. Operators can use the recovery image, which is included in the SR Linux software distribution package, to reimage the SSDs on the system.

This section describes the procedure for creating a recovery SD card containing the SR Linux software recovery image, which is used to reimage the SSDs on the 7250 IXR-X1b/X3b/X4 system.

Note: For recovery procedures on 7220 IXR-H4/H5/DL/H4-32D systems, see Performing USB recovery for SSD-boot platforms.

For starting SR Linux with factory defaults, see Starting SR Linux with factory defaults or post-ONIE installation.

5.6.1 Creating a recovery SD card

About this task

Note:
This procedure applies only to 7250 IXR-X1b/X3b/X4 platforms.

The 7220 IXR-H4/H5/DL/H4-32D platforms do not include an SD card slot and support recovery only through USB. For information on creating a recovery image for these platforms, see Creating a USB recovery image.

Creating a recovery SD card requires a working Linux system running Debian/Ubuntu (Debian version 12 or higher, Ubuntu version 20 or higher) or CentOS (version 7 or higher) with access to an SD card, preferably 16 GB. The SD card should be formatted and have no important data present on it. Any data on the card is wiped during the procedure.

Procedure

Step 1. Download the recovery image from the SR Linux software distribution package to the server being used to prepare the recovery disk.

Step 2. Open a terminal and navigate to the directory where you downloaded the recovery image.

Step 3. Extract the image using the following command:

Example

```
gunzip -c srlinux-24.7.1.bin.srlinuxrecoverydisk-img.gz > srlinux-
24.7.1.bin.srlinuxrecoverydisk-img
```

Step 4. Insert the SD card into the system to flash the recovery image.

Ensure the system correctly identifies the SD card because this action is destructive. The exact device path to your SD card depends on your Linux distribution and system setup. In this procedure, the SD card device is detected as `/dev/sdb`.

- If an SD card is connected through a USB adapter, the Linux system may identify it as `/dev/sdb` (with `/dev/sda` being a boot drive).
- If an SD card is connected to a device via an SD slot, the Linux system may identify it as `/dev/mmcblk0` (or `mmcblk1`, `mmcblk2`), depending on which MMC slot is used.

Note: If the system does not automatically detect the partition path, use the following commands to know the right partition path.

```
df -h
lsblk
```

If the correct path is unclear from the output, run the command both with and without the SD card inserted.

Step 5. Before flashing the SD card, unmount all partition paths. To find the partition path, run the command `df -h | grep /dev/sdb` and unmount all the partition paths displayed in the output using the following command:

`# umount <dev-path-to-sdcard-partitions>`

Example

```
# umount /dev/sdbX
```

Step 6. Flash the SD card with the recovery image file.

`sudo dd if=<machine>.srlinuxrecoverydisk-img of=/dev/sdX bs=4M status=progress`
where *machine* = the image name for the device and *sdX* = the USB device name.

Example

The following is a command execution example on a Debian system.

```
sudo dd if=srlinux-24.7.1.bin.srlinuxrecoverydisk-img of=/dev/sdX bs=4M status=
progress
```

Step 7. Umount the SD card. For instructions, see step 5.

Example

```
# umount /dev/sdbX
```

Step 8. Repeat steps 4 to 7 with another SD card for the standby control plane module (if applicable).

5.6.2 SSD reimaging from a recovery SD card

About this task

You can use the recovery SD card containing the SR Linux recovery image for SSD boot on 7250 IXR-X1b/X3b/X4 system. The 7220 IXR-H4/H5/DL/H4-32D systems support SSD reimaging only through a USB-based recovery image . For details, see SSD reimaging using a recovery USB drive.

WARNING: Reimaging the SSD removes all existing content.

Procedure

Step 1. Place the SD card containing an SR Linux recovery image into the appropriate external SD card slot of a 7250 IXR-X1b/X3b/X4. For detailed instructions, refer to the 7250 IXR-X Chassis Installation Guide.

Step 2. Connect to the console port and reboot the box while monitoring the console output.

Step 3. Press the Up arrow key when the Nokia banner appears to enter the BIOS boot menu.

Step 4. Select **USB HDD: Generic Ultra HS-COMBO** from boot options .

Step 5. Verify the Grub menu. If you see the following, continue to the next step. Otherwise, reboot into the BIOS menu and repeat step 3.

```
GNU GRUB version 2.06-13+srl1
+----------------------------------------------------------------------------+
| srlinux-rescue                                                             |
|                                                                            |
+----------------------------------------------------------------------------+
```

Step 6. SR Linux rescue automatically runs and installs `srlinux.bin` from the second partition of the SD card (`NOKIA-RECDISK`) onto the 7250 IXR-X1b/X3b/X4 system SSD.

Step 7. The 7250 IXR-X1b/X3b/X4 system reboot from the SSD with the updated SR Linux image. The SR Linux services and applications are automatically started. To boot the system with the SR Linux image, see Starting SR Linux with factory defaults or post-ONIE installation.

Step 8. Remove the recovery SD card from the system.

5.7 Performing USB recovery for SSD-boot platforms

On 7220 IXR-H4/H5/DL/H4-32D and 7250 IXR-X1b/X3b/X4 systems, Secure Boot is enabled by default, and the system boots from SSD. Operators can use the recovery image, which is included in the SR Linux software distribution package, to reimage the SSDs on the system.

The 7220 IXR-H4/H5/DL/H4-32D platforms do not include an SD card slot and support recovery only via USB. The following section describes the procedure for creating a recovery USB containing the SR Linux software recovery image, which is used to reimage the SSDs on a 7220 IXR-H4/H5/DL/H4-32D system.

Note: The 7250 IXR-X1b/X3b/X4 system includes both SD Card and USB and supports recovery only through the SD card. For recovery procedures, see Performing SD card recovery for SSD-boot platforms.

For starting SR Linux with factory defaults, see Starting SR Linux with factory defaults or post-ONIE installation.

5.7.1 Creating a USB recovery image

About this task

Note:
This procedure applies only to 7220 IXR-H4/H5/DL/H4-32D platforms. The 7250 IXR-X1b/X3b/X4 system supports recovery from the SD card. For information on creating a recovery SD card for the 7250 IXR-X1b/X3b/X4 system, see Creating a recovery SD card.

Installing a recovery image on a 7220 IXR-H4/H5/DL/H4-32D system requires a working Linux system and a USB device.

Procedure

Step 1. Download the recovery image from the SR Linux software distribution package to the server being used to prepare the recovery USB drive.

Step 2. Open a terminal and navigate to the directory where you downloaded the recovery image.

Step 3. Copy the recovery .iso image file to a USB using the following command:

`dd if=<machine>.iso of=/dev/sdX bs=10M`

Where *machine* = the image name for the device and *sdX* = the USB device name.

Example

The following is a command execution example on a Debian system.

```
sudo dd if=srlinux-25.10.1.bin.srlinuxrecoverydisk-img of=/dev/sdX bs=4M status=
progress
```

Step 4. After the recovery .iso image is copied, unmount the USB device and remove it from the Linux machine. For SSD reimaging from a USB, see SSD reimaging using a recovery USB drive.

5.7.2 SSD reimaging using a recovery USB drive

About this task

The 7220 IXR-H4/H5/DL/H4-32D systems support SSD reimaging only through a USB-based recovery image.

WARNING: Reimaging the SSD removes all existing content.

Procedure

Step 1. Insert a USB drive containing an SR Linux recovery image into the appropriate slot or port on a 7220 IXR-H4/H5/DL/H4-32D system. For detailed instructions, refer to the corresponding Chassis Installation Guide.

Step 2. Connect to the console port and reboot the box while monitoring the console output.

Step 3. Press the Up arrow key when the Nokia banner appears to enter the BIOS boot menu.

Step 4. **Note:** The boot menu entries may vary depending on the USB model number. The example shown uses a USB drive that is officially supported by Nokia.

Select **UEFI: Virtium TuffDrive 0508, Partition 1 (Virtium TuffDrive 0508)** from boot options.

Step 5. Verify the Grub menu. If you see the following, continue to the next step. Otherwise, reboot into the BIOS menu and repeat step 3.

```
GNU GRUB version 2.06-13+srl1
+----------------------------------------------------------------------------+
| srlinux-rescue                                                             |
|                                                                            |
+----------------------------------------------------------------------------+
```

Step 6. The system boots into SR Linux rescue recovery mode and installs the SR Linux image (`srlinux.bin`) from the NOKIA-RECDISK partition on the USB drive onto the SSD.

Step 7. After the image installation is complete, the system automatically reboots into the newly installed SR Linux. The SR Linux services and applications are automatically started. To boot the system with the SR Linux image, see Starting SR Linux with factory defaults or post-ONIE installation.

Step 8. Remove the USB drive from the system.

5.8 Bootstrapping using ONIE (7220 IXR-H series)

This section describes ONIE installation procedures applicable to all 7220 IXR-Dx/Hx platforms, **except** 7220 IXR-H4/H5/DL/H4-32D platforms. For recovery procedures on these systems, see Performing USB recovery for SSD-boot platforms.

For starting SR Linux with factory defaults, see Starting SR Linux with factory defaults or post-ONIE installation.

5.8.1 Image upgrade from ONIE prompt

About this task

This procedure applies only to the 7220 IXR-Dx/Hx platforms, excluding the 7220 IXR-H4/H5/DL/H4-32D platforms.

Note:
If you do not host the SR Linux images from a ZTP server, you must perform a manual bootstrapping to retrieve the image.

Procedure

Step 1. In the GRUB selection screen, select `ONIE`.

Step 2. From the list of ONIE boot options, select `ONIE: OS Install mode`.

Step 3. Enter the login credentials (when prompted by the system):

- username: `root`
- password: `NokiaSrl1!`

Step 4. After the ONIE image boots, the service discovery starts automatically. To stop the service discovery, execute:

```
ONIE:/ # onie-stop
```

Step 5. Configure the management IP address and the default route to copy the SR Linux image to the 7220 IXR system:

Example

```
ONIE:/ #
ONIE:/ # ifconfig eth0 135.227.251.182 netmask 255.255.255.0
ONIE:/ # ip route add 0.0.0.0/0 via 135.227.248.1
IP: RTNETLINK answers: Network is unreachable
ONIE:/ #
```

Step 6. Using the **SCP** command, copy the SR Linux image `<version>.bin` to the root folder.

```
# sudo scp <user>@<server-with-srlinux-image>:<path-to-srlinux-image-bin> <root>
```

Example:

```
# sudo scp serveruser@192.168.0.2:srlinux-24.10.1-459.bin /root
```

Step 7. To install SR Linux, execute the following command:

```
onie-nos-install <bin>
```

Example

```
ONIE:/ # onie-nos-install /root/srlinux-20.6.1-21398.bin
discover: installed mode detected.
Stopping: discover... done.
ONIE: Executing installer: /root/srlinux-20.6.1-21398.bin
/dev/console
Verifying archive integrity... 100%   MD5 checksums are OK. All good.
Uncompressing srlinux-20.6.1-21398  100%
Files used: srlinux-20.6.1-21398.squashfs, initramfs-4.18.39-2.x86_64-02.img, vmlinuz-
4.19.39-2.x86_64
Found ONIE-BOOT on /dev/sda2
Will use /dev/sda as install dev
Parts used: old_part_start[4], efi_part[4], nos_part[5], etc_part[6], opt_part[7],
 data_part[8]
Remove existing partitions from /dev/sda
/dev/sda4 is not mounted
Warning: The kernel is still using the old partition table.
The new table will be used at the next reboot.
The operation has completed successfully.
```

Step 8. To boot the system with the SR Linux image, see Starting SR Linux with factory defaults or post-ONIE installation.

5.8.2 Installing an ONIE image

About this task

Note:
This procedure applies only to the 7220 IXR-Dx/Hx platforms, excluding the 7220 IXR-H4/H5/DL/H4-32D platforms.

Installing an ONIE image on a 7220 IXR system requires a working Linux system and a USB device. Installation also requires the ONIE boot loader install environment.

WARNING: Installing the ONIE from the USB wipes out all SSD partitions.

Procedure

Step 1. Request the ONIE recovery .iso image for the respective 7220 IXR system from OLCS.

Step 2. Copy the ONIE recovery .iso image file to a USB using the following command:
`dd if=<machine>.iso of=/dev/sdX bs=10M`
where *machine* = the image name for the device and *sdX* = the USB device name.

Step 3. After the ONIE recovery .iso image is copied, unmount the USB device and remove it from the Linux machine.

Step 4. Insert the USB into the 7220 IXR system and power the system on.

Step 5. When the setup message comes up, press either the **DEL** or **ESC** key to enter the BIOS interface:

```
Version 2.19.1266. Copyright (C) 2019 American Megatrends. Inc.
BIOS Date: 11/01/2019 15:48:23 Ver: OACHI037 Minor_Ver: V1.03
Press <DEL> or <ESC> to enter setup.
Entering Setup...
```

Step 6. In the BIOS prompt, select **Boot Device as USB**, then **Save & Exit**.

Step 7. Install the ONIE from the USB. Select **ONIE: Embed ONIE** in the GNU Grub screen.

Step 8. After the ONIE installation is complete, remove the USB to boot the ONIE from the SSD.

Step 9. After the device boots the ONIE from the SSD, select **ONIE: Install OS** in the GNU Grub screen.

Step 10. Verify the platform, version, and build date of the installed ONIE image:

```
GRUB loading.
Welcome to GRUB!

Platform  : x86_64-nokia_ixr7220_d3-r0
Version   : 2019.02-onie_version-v1.5
Build Date: 2020-02-13T15:05+08:00

telnet>
```

Step 11. The device boots and enters the `ONIE:/ #` prompt.

The ONIE service discovery automatically gets a device IP address from a ZTP server, and the SR Linux image is downloaded.

Note: If you do not host the SR Linux images from a ZTP server, you must perform a manual bootstrap procedure to complete the installation. See the Image upgrade from ONIE prompt procedure to continue.

Step 12. After the SR Linux software installation completes, the 7220 IXR reboots with the updated SR Linux image. The SR Linux services and applications are automatically started.

5.9 Bootstrapping using ONIE (7215 IXS system)

This section describes ONIE installation procedures applicable to 7215 IXS system.

For starting SR Linux with factory defaults, see Starting SR Linux with factory defaults or post-ONIE installation.

5.9.1 Image upgrade from ONIE prompt

About this task

This procedure applies only to the 7215 IXS platforms.

Note:
If you do not host the SR Linux images from a ZTP server, you must perform a manual bootstrapping to retrieve the image.

Procedure

Step 1. Power on the 7215 IXS-A1 system while monitoring the console output.

Step 2. Interrupt the normal boot process by pressing **CTRL+C** at this prompt:

Note: When prompted, enter password `NokiaSrl1!` to proceed.

```
Model: Nokia 7215 IXS-A1
CPLD:  v14
Net:   eth0: mvpp2-2 [PRIME]

AC5X Init  ... done

BusDevFun  VendorId   DeviceId   Device Class       Sub-Class
_____________________________________________________________
00.00.00   0x11ab     0x0110     Bridge device           0x04
01.00.00   0x11ab     0x9821     Network controller      0x00

PCIe Init ... done
AC5X MPP pins Init ... done
OOB MGNT PHY Init Dn ... done
CP GPIO pins Init ... done
Autoboot in 5 seconds, to stop use 'CTRL-C'
Enter passwd> **********
```

```
OOB MGNT PHY Init Up ... done
Nokia>>
```

Step 3. After entering the password as mentioned in step 2, execute the following command at the Nokia prompt, to initiate the ONIE process.

```
Nokia>> run onie_boot
```

Step 4. After the ONIE image boots, the service discovery starts automatically. To stop the service discovery, execute the following command:

```
ONIE:/ # onie-stop
```

Step 5. Configure the management IP address and the default route to copy the SR Linux image to the 7215 IXS-A1 system:

Example

```
ONIE:/ #
ONIE:/ # ifconfig eth0 135.227.251.182 netmask 255.255.255.0
ONIE:/ # ip route add 0.0.0.0/0 via 135.227.248.1
IP: RTNETLINK answers: Network is unreachable
ONIE:/ #
```

Step 6. Using the **SCP** command, copy the SR Linux image <version>.bin to the root folder.

 # sudo scp <user>@<server-with-srlinux-image>:<path-to-srlinux-image-bin> <root>

Example

```
# sudo scp serveruser@192.168.0.2:srlinux-24.10.1-459.bin /root
```

Step 7. To install SR Linux, execute the following command:

onie-nos-install <bin>

Example

```
ONIE:/ # onie-nos-install /root/srlinux-24.10.1-248.bin
discover: installer mode detected.
Stopping: discover... done.
ONIE: Executing installer: /root/srlinux-24.10.1-248.bin
Verifying archive integrity...  100%   MD5 checksums are OK. All good.
Uncompressing srlinux-24.10.1-248:release  100%
Files used: srlinux_fit-6.1.25-28-arm64.itb _srlinux_fit_rescue-6.1.25-28-arm64-
41.itb_ srlinux-24.10.1-248.squashfs
Found mmc0:0001 on /dev/mmcblk0
Will use /dev/mmcblk0 as install dev
Remove existing partitions except ONIE from /dev/mmcblk0
/dev/mmcblk0p2 is not mounted
Warning: The kernel is still using the old partition table.
The new table will be used at the next reboot.
The operation has completed successfully.
/dev/mmcblk0p3 is not mounted
Warning: The kernel is still using the old partition table.
The new table will be used at the next reboot.
The operation has completed successfully.
```

```
/dev/mmcblk0p4 is not mounted
Warning: The kernel is still using the old partition table.
The new table will be used at the next reboot.
The operation has completed successfully.
/dev/mmcblk0p5 is not mounted
Warning: The kernel is still using the old partition table.
The new table will be used at the next reboot.
The operation has completed successfully.
/dev/mmcblk0p6 is not mounted
Warning: The kernel is still using the old partition table.
The new table will be used at the next reboot.
The operation has completed successfully.
/dev/mmcblk0p7 is not present
/dev/mmcblk0p8 is not present
/dev/mmcblk0p9 is not present
Start creating new partitions on /dev/mmcblk0
Create partition part:2, size:200, label:EFI-System, typecode:ef02
Warning: The kernel is still using the old partition table.
The new table will be used at the next reboot.
The operation has completed successfully.
Create partition part:3, size:5000, label:NOKIA-OS, typecode:8300
Warning: The kernel is still using the old partition table.
The new table will be used at the next reboot.
The operation has completed successfully.
Create partition part:4, size:1024, label:NOKIA-ETC, typecode:8300
Warning: The kernel is still using the old partition table.
The new table will be used at the next reboot.
The operation has completed successfully.
Create partition part:5, size:3000, label:NOKIA-OPT, typecode:8300
Warning: The kernel is still using the old partition table.
The new table will be used at the next reboot.
The operation has completed successfully.
Create partition part:6, size:-1, label:NOKIA-DATA, typecode:8300
Warning: The kernel is still using the old partition table.
The new table will be used at the next reboot.
The operation has completed successfully.

Disk /dev/mmcblk0: 21.2 GB, 21206401024 bytes
256 heads, 63 sectors/track, 2568 cylinders
Units = cylinders of 16128 * 512 = 8257536 bytes

        Device Boot      Start         End      Blocks  Id System
/dev/mmcblk0p1               1        2569    20709375+ ee EFI GPT
Creating file systems
Fat32 fs for EFI-System
Ext4 fs for NOKIA-OS
mke2fs 1.46.3 (27-Jul-2021)
Discarding device blocks: done
Creating filesystem with 1280000 4k blocks and 320000 inodes
Filesystem UUID: 99791b85-5eb5-4ed1-9e45-f9b545f3c1cb
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done

Ext4 fs for NOKIA-ETC
mke2fs 1.46.3 (27-Jul-2021)
Discarding device blocks: done
Creating filesystem with 262144 4k blocks and 65536 inodes
Filesystem UUID: 81d1c5e6-16bb-43d2-9388-52811569241f
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376
```

```
Allocating group tables: done
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

Ext4 fs for NOKIA-OPT
mke2fs 1.46.3 (27-Jul-2021)
Discarding device blocks: done
Creating filesystem with 768000 4k blocks and 192000 inodes
Filesystem UUID: 62993828-5e67-4a76-9a5f-e6d7fa9d59f5
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done

Ext4 fs for NOKIA-DATA
mke2fs 1.46.3 (27-Jul-2021)
Discarding device blocks: done
Creating filesystem with 1986048 4k blocks and 496784 inodes
Filesystem UUID: 7877d117-d508-4696-8e2f-347787b101a5
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done

Copy images to install dev
Copying FIT image from /tmp/selfgz102031668/srlinux_fit-6.1.25-28-arm64.itb to /tmp/
srlsd-nos-XXXXbOkeeu/24.10.1-248/
FIT image copied
Copying squashfs from /tmp/selfgz102031668/srlinux-24.10.1-248.squashfs to /tmp/srlsd-
nos-XXXXbOkeeu/24.10.1-248/
squashfs copied
Copy rescue image
Copying rescue FIT image from /tmp/selfgz102031668/_srlinux_fit_rescue-6.1.25-28-
arm64-41.itb_ to /tmp/srlsd-rescue-XXXXgcOb8t/rescue/srlinux_fit_rescue-6.1.25-28-
arm64-41.itb
Copy rescue image completed
umounting /tmp/srlsd-nos-XXXXbOkeeu
U-Boot Environment variables updated.
Perform final clean up
Srlinux installation completed
ONIE: NOS install successful: /root/srlinux-24.10.1-248.bin
ONIE: Rebooting...
```

Step 8. To boot the system with the SR Linux image, see Starting SR Linux with factory defaults or post-ONIE installation.

5.9.2 Installing an ONIE image

About this task

Note: This procedure should only be executed if the internal disk partitions are corrupted or missing.

Installing an ONIE image on a 7215 IXS system requires a working Linux system and a USB device formatted as FAT 32. Installation also requires the ONIE boot loader install environment.

WARNING: Installing the ONIE from the USB wipes out all SSD partitions.

Procedure

Step 1. Request the ONIE recovery FIT image (nokia_ixs7215_52xb-r0.itb) for the 7215 IXS-A1 system from OLCS.

Step 2. Mount the USB and copy the ONIE recovery FIT image file to a USB using the following command:

```
$ sudo mount /dev/sda1 /media/SANDISK-SSD
$ sudo cp /tmp/nokia_ixs7215_52xb-r0.itb /media/SANDISK-SSD/
$ sync
$ sudo umount  /media/SANDISK-SSD
```

Step 3. After the ONIE recovery FIT image is copied, unmount the USB device and remove it from the Linux machine.

Step 4. Insert the USB into the 7215 IXS-A1 system and power on the system.

Step 5. Interrupt the normal boot process by pressing **CTRL+C** at this prompt:

Note: When prompted, enter password `NokiaSrl1!` to proceed.

```
Model: Nokia 7215 IXS-A1
CPLD:  v14
Net:   eth0: mvpp2-2 [PRIME]

AC5X Init  ... done

BusDevFun  VendorId   DeviceId   Device Class       Sub-Class
_____________________________________________________________
00.00.00   0x11ab     0x0110     Bridge device           0x04
01.00.00   0x11ab     0x9821     Network controller      0x00

PCIe Init ... done
AC5X MPP pins Init ... done
OOB MGNT PHY Init Dn ... done
CP GPIO pins Init ... done
Autoboot in 5 seconds, to stop use 'CTRL-C'
Enter passwd> **********
OOB MGNT PHY Init Up ... done
Nokia>>
```

Step 6. After entering the password as mentioned in step 5, execute the following command at the Nokia prompt, to initiate the USB port.

```
Nokia>> usb start
```

Example

```
Nokia>> usb start
```

```
OOB MGNT PHY Init Dn ... done
CP GPIO pins Init ... done
Autoboot in 5 seconds, to stop use 'CTRL-C'
Enter passwd> **********
OOB MGNT PHY Init Up ... done
Nokia>> usb start
starting USB...
Bus usb3@500000: Register 2000120 NbrPorts 2
Starting the controller
USB XHCI 1.00
Bus usb3@510000: Register 2000120 NbrPorts 2
Starting the controller
USB XHCI 1.00
scanning bus usb3@500000 for devices... 2 USB Device(s) found
scanning bus usb3@510000 for devices... 1 USB Device(s) found
       scanning usb for storage devices... 1 Storage Device(s) found
Nokia>>
```

Step 7. Verify whether the ONIE FIT file is on the USB disk by executing the following command at the Nokia prompt.

```
Nokia>> fatls usb 0
```

Example

```
Nokia>> fatls usb 0
            System Volume Information/
11855404   nokia_ixs7215_52xb-r0.itb
   935364   u-boot-a38x-Nokia-7215-spi.bin

2 file(s), 1 dir(s)
Nokia>>
```

Step 8. Load the image by executing the following command:

```
Nokia>> fatload usb 0 0x800000 nokia_ixs7215_52xb-r0.itb
```

Example

```
Nokia>> fatload usb 0 0x800000 nokia_ixs7215_52xb-r0.itb
11855404 bytes read in 557 ms (20.3 MiB/s)
Nokia>>
```

Step 9. Execute the following commands separately from the Nokia prompt to set the boot parameters.

```
Nokia>> setenv console "console=ttyS0,115200 earlycon=uart8250,mmio32,0xF0512000"
Nokia>> setenv bootargs "root=/dev/ram0 rw debug panic=1 $console quiet"
Nokia>>
```

Step 10. Boot the ONIE image.

```
bootm 0x800000
```

Step 11. After the ONIE image boots, the service discovery starts automatically. To stop the service discovery, execute the following command:

```
ONIE:/ # onie-stop
```

Step 12. At this point, ONIE should automatically create the required partitions, including the ONIE and swap partitions.

a. Verify the partition contents.

```
ONIE:/ # sgdisk -p /dev/mmcblk0
Disk /dev/mmcblk0: 41418752 sectors, 19.8 GiB
Logical sector size: 512 bytes
Disk identifier (GUID): C1052BC4-9C19-4048-8B4E-3E6EDC61BAC3
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 41418718
Partitions will be aligned on 2048-sector boundaries
Total free space is 34781150 sectors (16.6 GiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048          346111   168.0 MiB   8300  ONIE
  10        35125248        41418718   3.0 GiB     8200  swap
```

Step 13. Copy the FIT file from the USB disk to the internal ONIE partition.

a. Mount the internal ONIE partition (`mmcblk0p1`).

```
ONIE:/ # mkdir /mnt/p1
ONIE:/ # mount /dev/mmcblk0p1 /mnt/p1/
```

b. Mount the USB disk (`sda1`).

Example

```
ONIE:/ # mkdir /mnt/usb/
```

```
ONIE:/ # mount /dev/sda1 /mnt/usb/
```

Note: The directory `/mnt/usb/` should already exist; if it does not, create it using the command:

```
mkdir -p /mnt/usb
```

c. Copy the FIT file.

Example

```
ONIE:/ # cp /mnt/usb/nokia_ixs7215_52xb-r0.itb /mnt/p1/
ONIE:/ # ls -l /mnt/p1/
drwx------    2 root     0            12288 Oct 16 12:00 lost+found
-rwxr-xr-x    1 root     0         11855404 Oct 16 12:13 nokia_ixs7215_52xb-r0.itb
```

Step 14. Reboot the system.

Example

```
ONIE:/ # sync
```

```
ONIE:/ # reboot
```

Step 15. To enter ONIE and install the SR Linux image after rebooting, see the Image upgrade from ONIE prompt.

6 Zero Touch Provisioning

This chapter describes Zero Touch Provisioning (ZTP) on SR Linux. Traditional deployment of new nodes in a network is a multistep process where the user has to connect to the hardware and provision global and local parameters.

ZTP automatically configures the nodes by obtaining the required information from the network and provisioning them with minimal manual intervention and configuration. The technician installs the nodes into the rack and when power is applied, and if connectivity is available, the nodes are auto-provisioned.

6.1 Applicability

The following implementation is currently supported:

- auto-boot using the Out-of-Band (OOB) port, available on all 7250 IXR routers, supports HTTP, HTTPS, TFTP, and FTP
- auto-boot using the in-band port, available only on these platforms, supports HTTP, HTTPS, TFTP, and FTP
  - 7220 IXR-D1
  - 7220 IXR-D4
  - 7220 IXR-D5
  - 7220 IXR-DL
  - 7215 IXS-A1
- VLAN discovery is supported on these platforms only through the in-band management port and is not available on out-of-band management ports
  - 7220 IXR-D1
  - 7220 IXR-DL
  - 7215 IXS-A1
- Out-of-band management supports dual-stack IPv4 and IPv6, including DHCP IPv4/IPv6 client
- In-band management port supports only IPv4, including IPv4 DHCP client

For details on operational guidelines that apply to in-band and out-of-band management ports, see Out-of-band management versus in-band management.

6.2 ZTP overview

The auto-boot process can use the out-of-band (mgmt port) or in-band management on Ethernet ports. The node storage device ships with the SR Linux image and the grub.cfg for auto-boot using the out-of-band port or the in-band port; within the `grub.cfg` is an auto-boot flag. The auto-boot flag is enabled by default and can be manually changed if needed.

When SR Linux boots, it checks the grub.cfg for the auto-boot flag. As the flag is set by default, the node enters auto-boot mode.

The node attempts the auto-boot process using any port with an operational link, starting with the out-of-band management port. If the node cannot be discovered using the out-of-band management port, either because the port is down or is not receiving a DHCP offer from the DHCP server, the process is reattempted using the in-band Ethernet ports. If the in-band Ethernet ports fail, the process then attempts VLAN discovery. ZTP VLAN discovery is enabled by default during the in-band process, allowing the node to attempt discovery over predefined VLANs if required. The DHCP discovery can also be flooded on all available VLANs, including VLAN0, to speed up the in-band auto-discovery process. If the VLAN discovery fails, the out-of-band management port is tried again, and the cycle repeats sequentially.

When initiated, the auto-boot mode starts the auto-provisioning process. The auto-provisioning process discovers the IP address of the node and provisions the node based on a Python provisioning script.

The DHCP server provides the node with the location of the provisioning script using either the IPv4 DHCP Options 66 and 67, or Option 67, or Option 43. The node uses this URL to download the provisioning script. The provisioning script contains the location of the RPMs, configurations, images, and scripts. These files are downloaded to the storage device.

Secure ZTP (sZTP) is automatically selected when the DHCP server returns secure options (`OPTION_V4_SZTP_REDIRECT(143)` or `OPTION_V6_SZTP_REDIRECT(136)`), providing URIs for secure bootstrap servers. For information about DHCP server options based ZTP selection, see DHCP option based ZTP selection.

During provisioning, all events are logged and displayed at the console for debugging. After the process completes, the auto-boot flag is removed from the grub.cfg file. This ensures that after a successful auto-boot, additional reboots of the node do not enter auto-boot mode.

6.2.1 Network requirements

ZTP requires the following components:

- DHCP server (IPv4 or IPv6) – To support the assignment of IP addresses through DHCP requests and offers.
- file server – For staging and transfer of RPMs, configurations, images, and scripts. HTTP, HTTPS, TFTP, and FTP are supported. For HTTPS, the default Mozilla certificate should be used.
- DHCP relay – Required if the server is outside the management interface broadcast domain.

ZTP works in the following network environments:

- nodes, HTTP file servers, and DHCP server in the same subnet
- HTTP file servers and DHCP server in the same subnet, separate from the nodes
- nodes, HTTP file servers, and DHCP server in different subnets

Figure 2: All components in the same subnet shows the first scenario where all components are in a Layer 2 broadcast domain. There is no DHCP relay and all IPs are assigned from a single pool.

Figure 2: All components in the same subnet

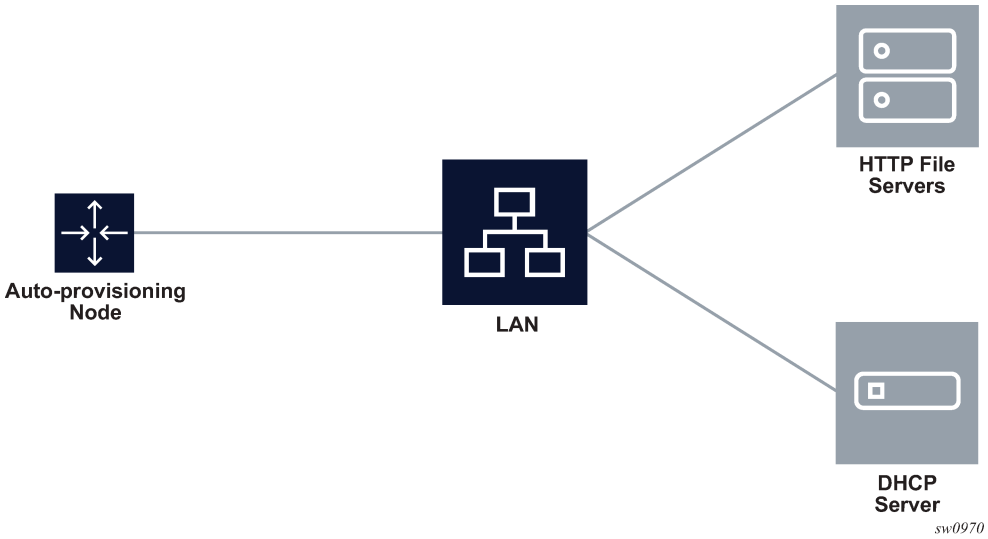


Figure 3: HTTP file and DHCP servers in the same subnet shows the second scenario where only the HTTP file servers and DHCP server are in the same subnet. The DHCP relay is used to fill Option 82 as the gateway address. The gateway address is used to find the appropriate pool in the DHCP server to assign the correct subnet IP address to the SR Linux system.

The DHCP offer allows the Option 3 router to define the default gateway. If multiple addresses are provided via Option 3, the first address is used for the default gateway.

Figure 3: HTTP file and DHCP servers in the same subnet

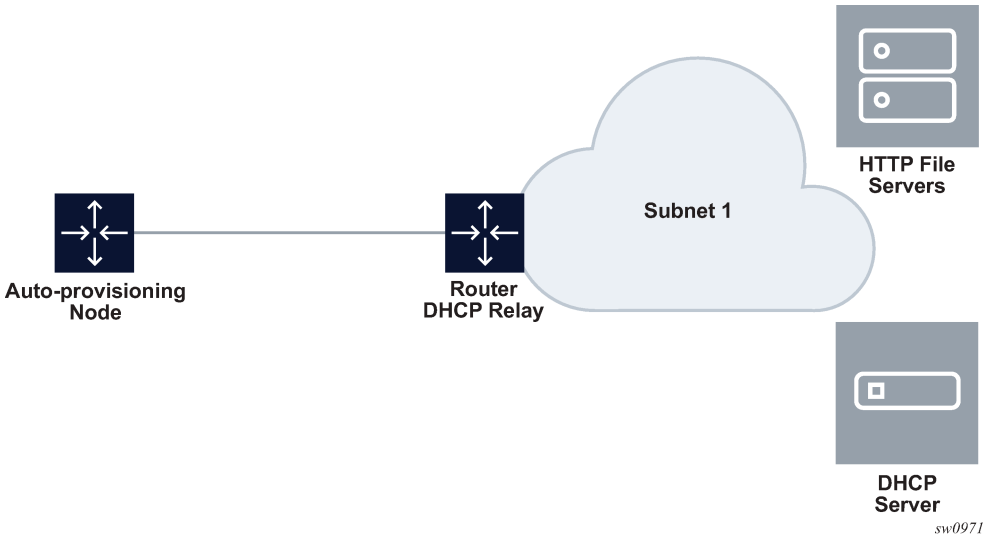
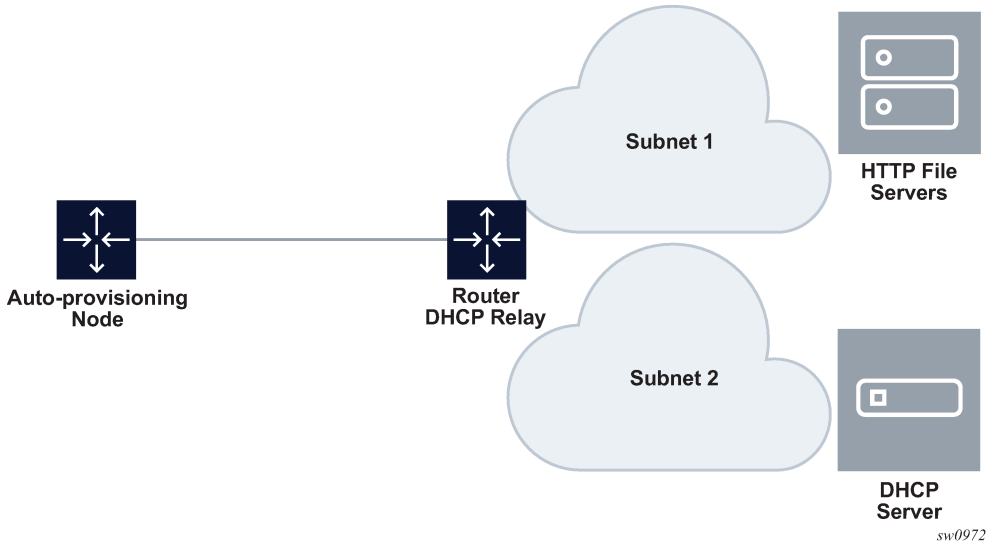


Figure 4: All components in different subnets shows the third scenario where all components are in different subnets. The DHCP relay adds the Option 82 gateway address to the DHCP request, and the DHCP server adds the Option 3 with the gateway address of the HTTP file server.

Figure 4: All components in different subnets



6.3 Process information

When the node reboots, SR Linux starts the auto-boot process if the auto-boot flag is set in the grub.cfg.

Out-of-band management versus in-band management

The auto-boot process can use the out-of-band management port or in-band management on Ethernet ports.

The following operational guidelines apply to in-band and out-of-band management ports.

- Out-of-band management supports auto-boot using an out-of-band port
- In-band management supports auto-boot on in-band management on Ethernet ports
- Out-of-band management and in-band management support HTTP, HTTP, TFTP, and FTP protocols.
- Out-of-band management and in-band management support untagged frames.
- Out-of-band management support dual-stack IPv4 and IPv6.
- In-band management supports only IPv4.
- Out-of-band management support DHCP clients for IPv4 and IPv6.
- In-band management supports IPv4 DHCP clients only.
- Out-of-band management does not support dot1q (VLAN tags).
- In-band management supports dot1q interfaces if the VLAN discovery option is correctly configured in the `grub.cfg` file.
- In-band ports support VLAN discovery for IPv4 by default if not disabled in the `grub.cfg` file.

6.3.1 DHCP discovery and solicitation

IPv4 DHCP discovery messages are sent from out-of-band and in-band management ports with active links, while IPv6 DHCP solicitation messages are only sent from the out-of-band ports.

For out-of-band ports:

1. IPv4 DHCP discovery with untagged frames is sent from the management port that is operationally up.
2. If the link is operationally down or there is no DHCP offer within the DHCP timeout, an IPv6 DHCP solicitation is sent out.
3. If neither IPv4 nor IPV6 DHCP offers are received, the system uses the in-band Ethernet ports for the auto-boot process.

For in-band ports:

1. IPv4 DHCP discovery frames is sent from each operationally up in-band port.
2. If there is no DHCP offer within the DHCP timeout, the DHCP discovery process is repeated sequentially for each in-band port, until a DHCP offer is successfully received.

- For DHCP IPv4, Option 61 is used for pool selection. By default, the node sends Option 61 with the serial number of the chassis.

  Note: The grub.cfg can be provisioned with a MAC option as well. When a MAC option is specified, Option 61 is populated with the chassis MAC address.

- For DHCP IPv6, Option 1 is used for pool selection. By default, the node uses RFC 3315 DUID Type 2 vendor-assigned unique ID. The value for *enterprise-id* is 6527 and the identifier is the chassis serial number.

  Note: Type 3 is configurable in the grub.cfg.

When the DHCP server receives the discovery packet, it assigns the IP address to the node. The DHCP offer for IPv4 requires the options shown in Table 7: Required DHCP offer options.

Table 7: Required DHCP offer options

Option	Name	Description
viaddr	Client-Ip-Address	Network interface – IP address (for network consistency, a fixed IP address is recommended vs randomly assigned from the DHCP server IP pool)
1	Subnet Mask	Network interface – Subnet mask
3	Router	Network interface – Default gateway (Only the first router is used. Additional routers are ignored.)
51	Lease Time	Validated to be infinite
54	Server Address	DHCP server identifier
66	Boot server hostname	Server IP address

Option	Name	Description
67	Bootfile Name	URL or IP to the provisioning file

Table 8: DHCP IPv4 and IPv6 equivalents lists DHCP IPv4 and IPv6 equivalents.

Table 8: DHCP IPv4 and IPv6 equivalents

Option	IPv4 option	IPv6 Option	IPv6 Comments
Client ID	Option 61	Option 1 (DUID)	2 – Vendor-assigned unique ID (default) 3 – Link-layer address
NTP server	Option 42	Option 56	—
User class	Option 77	Option 15	—
TFTP server name	Option 66	NA	—
Bootfile name	Option 67	Option 59	—
Vendor-specific options	Option 43	Option 17	—

6.3.1.1 Auto-provisioning options

Defined options determine how DHCP discovery functions.

The client ID used in IPv4 and IPv6 can be configured as a chassis serial ID or chassis MAC address. By default, the chassis serial ID is used, but the user can configure the auto-boot option to use the chassis MAC address. This option can be configured by editing the grub.cfg and adding the `-mac` sub-option to the auto-boot option:

`grub.cfg` location: `/mnt/nokiaboot/boot/grub2/grub.cfg` or `/mnt/nokiaboot/boot/grub/grub.cfg`

The ZTP timeout default is set at 1 hour for each attempt. During the provisioning process, the node performs three attempts for a period of 3 hours (1 hour for each attempt). After the three initial attempts, the node reboots. The user can change the 1 hour default using the ZTP CLI.

See sections Configuring ZTP and ZTP CLI and SR Linux CLI command structures for procedures and commands used to provision available options.

6.3.1.2 DHCP server Option 42 (IPv4) and 56 (IPv6) for NTP

DHCP provides an NTP server URL using Option 42 (for IPv4) or Option 56 (for IPv6). When the time is obtained and synchronized from the NTP server, any log event obtained after this time has the correct timestamp. Any log event before the time was obtained does not have the correct timestamp, and has the default timestamp instead.

6.3.2 VLAN discovery

The node can perform VLAN discovery when shipped in ZTP mode. VLAN discovery is supported only for the in-band management port and is not available for the out-of-band management port.

Note: VLAN discovery is available only on 7220 IXR-D1, 7220 IXR-DL, and 7215 IXS-A1 platforms.

After the node is installed and powered up:

1. Initially, out-of-band ZTP is attempted only on the management interface, then in-band ZTP on all front panel ports.
   - SR Linux scans each port with an operational link and performs the following on all operational ports:
     a. sends the DHCP discovery request without a dot1q tag (untagged)
     b. sends the DHCP discovery request with a dot1q tag from the range of 0 to 4049
   - SR Linux waits for a DHCP offer response within the configured DHCP timeout.
2. The first VLAN with a valid offer that includes either the IPv4 DHCP Options 66 and 67, or Option 67, or Option 43, is selected as the working VLAN, and the ZTP process is executed on this VLAN.

   Note: If no DHCP offer is received or if the offer lacks the required options, SR Linux floods the network with DHCP discovery messages on all remaining non-reserved VLANs (1 to 4094).

3. When a VLAN is discovered, the ZTP process is executed on the respective VLAN.

   Note: If there is no offer on any VLAN or the offer does not have the correct options, the node starts over from step 1.

VLAN discovery option

By default, the auto-boot flag in the `grub.cfg` file has the VLAN discovery option enabled. The option can be disabled manually in the `grub.cfg` file.

`grub.cfg` location: `/mnt/nokiaboot/boot/grub2/grub.cfg` or `/mnt/nokiaboot/boot/grub/grub.cfg`

6.3.3 DHCP offer

Both in-band and out-of-band auto-boot processes support IPv4 discovery, while only the out-of-band supports IPv6 solicitation. When a DHCP offer is received, the provisioning script follows the same address family format for file download as the DHCP offer.

The first DHCP offer received with the correct Option 66 and 67 or Option 43 is used. The Python provisioning script is downloaded from the location defined by Option 66 and 67 or Option 43.

With an IPv4 DHCP offer, the node IP address and other information, such as the default route, is included.

For IPv6, only the IP arrives from the DHCP server. This offer does not include a prefix, and when it is received, the route is installed with a prefix of /128. This means that the prefix received from the RA is

ignored, and the node always acts as a host with a prefix of /128. The default route is received from the Route Advertisement (RA) from the IPv6 peer.

6.3.3.1 Default gateway route configuration for IPv4

After the DHCP offer is received, the Option 3 router can be used to program the default gateway on the SR Linux. A static route should be configured with the default route 0.0.0.0/0 as the next-hop IP address (provided by the Option 3 router).

6.3.3.2 DHCP relay

The DHCP relay can be used to fill Option 82 for the gateway address. The gateway address can be used to find the appropriate pool in the DHCP server to assign the correct subnet IP address to the SR Linux.

6.3.4 Python provisioning script

The Python provisioning file is downloaded from the location dictated by Option 66 and 67 or 43 using HTTP, HTTPS, TFTP, or FTP. Option 66 and 67 takes precedence over Option 43 when both are present in the offer, but Option 43 is used if there is a download error with Option 66 and 67. In addition, Option 66 and 67 can be summarized in Option 67 only. The URL of the provisioning file can be resolved via the DHCP-provided DNS. Up to three DNS servers can be offered by the DHCP.

The node downloads the Python script to storage device. The node then uses the Python provisioning script to download any Debian packages, images, scripts, or configurations to the destination dictated by the script.

After successful completion, the provisioning script disables the auto-boot flag to ensure that additional reboots of the node do not enter auto-boot mode. When the nodes reboot, they come up in an operational state with the configuration and image.

For more information about the Python provisioning script, see Configuring the Python provisioning script.

6.3.5 Auto-provisioning failures

The following are possible failure scenarios:

- No Option 66 and 67 or 43 is received (possibly because the format is a URL or no IP address was provided via the DHCP server).
- The download of the Python provisioning script failed or the server was not reachable.
- Copying the Debian packages, images, scripts, or configuration to the storage device failed.

If a failure occurs:

- Details of the failure display on the console and are recorded in the appropriate log files. Log files are stored in the storage device. There can be three log files, which are overwritten in a circular manner.
- The DHCP task is notified of the failure and releases the IP address on the port.
- The auto-boot task goes through the process cycle again until it succeeds, the timeout value is reached, or the auto-boot Option is removed from the grub.cfg, either by editing the grub.cfg or using the ZTP CLI.

6.3.6 ZTP log files

ZTP log files are stored under `/var/log/ztp`.

Example:

```
[root@srlinux ztp]# ls -ltr
total 28528
-rw-r--r-- 1 root root    18220 Sep  4 23:04 ztp_2019-09-04_23-03-52_880867.log
-rw-r--r-- 1 root root     1789 Sep  4 23:29 ztp_2019-09-04_23-29-38_496995.log
-rw-r--r-- 1 root root    18220 Sep  4 23:31 ztp_2019-09-04_23-31-08_996284.log
-rw-r--r-- 1 root root     1791 Sep  5 17:42 ztp_2019-09-05_17-42-01_082002.log
-rw-r--r-- 1 root root        0 Sep  5 21:56 ztp_2019-09-05_21-56-13_730783.log
-rw-r--r-- 1 root root        0 Sep  5 21:56 ztp_2019-09-05_21-56-14_036070.log
[root@srlinux ztp]#
```

6.4 Configuring ZTP

The following are common ZTP configuration procedures:

- Configuring the Python provisioning script
- Configuring the ZTP timeout value using the provisioning script

The following are common ZTP configuration procedures using the ZTP CLI:

- Configuring options in the grub.cfg using ZTP CLI
- Managing images using ZTP CLI
- Configuring the NOS using ZTP CLI
- Redownloading the executable files with ZTP CLI
- Starting, stopping, and restarting a ZTP process using ZTP CLI
- Checking the status of a ZTP process using ZTP CLI

The following are common ZTP configuration procedures using the SR Linux CLI:

- Configuring options in the grub.cfg using SR Linux CLI
- Specifying the image, kernel, or RAM to boot the system using SR Linux CLI
- Starting, stopping, and restarting a ZTP process using SR Linux CLI
- Checking the status of a ZTP process using SR Linux CLI

6.4.1 ZTP CLI versus SR Linux CLI

There are two CLIs where ZTP-related commands can be executed:

- SR Linux CLI (sr_cli)
- ZTP CLI

The SR Linux commands are available to use only when the SR Linux is operational at the sr_cli.

When the SR Linux is not operational, the ZTP CLI is a unified tool that can be used to manage ZTP tasks at the console.

See the ZTP CLI and SR Linux CLI command structures sections for a complete list of available ZTP-related commands.

6.4.2 Configuring the Python provisioning script

The Python provisioning script may include the following components:

- location of provider certificate and trust anchor when HTTPS is used to download Debian packages and other Bash/Python scripts
- the installation of Debian packages, location of script, and configuration
- a section to clear the auto-boot option from the grub.cfg kernel section

Example: Python provisioning script (Out-of-Band ZTP)

```
import errno
import os
import sys
import signal
import subprocess
from subprocess import Popen, PIPE
import threading
srlinux_image_url = 'http://135.227.249.116/srlinux/srlinux-20.6.1-10656.squashfs'
srlinux_image_md5_url = 'http://135.227.249.116/srlinux/srlinux-20.6.1-10656.md5'
srlinux_config_url = 'http://135.227.249.116/srlinux/config.json'
class ProcessError(Exception):
    def __init__(self, msg, errno=-1):
        Exception.__init__(self, msg)
        self.errno = errno
class ProcessOpen(Popen):
    def __init__(self, cmd, cwd=None, env=None, flags=None, stdin=None,
        stdout=None, stderr=None, universal_newlines=True,):
        self.__use_killpg = False
        shell = False
        if not isinstance(cmd, (list, tuple)):
            shell = True
        # Set flags to 0, subprocess raises an exception otherwise.
        flags = 0
        # Set a preexec function, this will make the sub-process create it's
        # own session and process group - bug 80651, bug 85693.
        preexec_fn = os.setsid
        self.__cmd = cmd
        self.__retval = None
        self.__hasTerminated = threading.Condition()
        Popen.__init__(self, cmd, cwd=cwd, env=env, shell=shell, stdin=stdin,
           stdout=PIPE, stderr=PIPE, close_fds=True,
           universal_newlines=universal_newlines, creationflags=flags,)
        print("Process [{}] pid [{}]".format(cmd, self.pid))
    def _getReturncode(self):
        return self.__returncode
    def __finalize(self):
        # Any finalize actions
        pass
    def _setReturncode(self, value):
        self.__returncode = value
        if value is not None:
            # Notify that the process is done.
            self.__hasTerminated.acquire()
            self.__hasTerminated.notifyAll()
            self.__hasTerminated.release()
    returncode = property(fget=_getReturncode, fset=_setReturncode)
```

```
    def _getRetval(self):
        # Ensure the returncode is set by subprocess if the process is finished.
        self.poll()
        return self.returncode
    retval = property(fget=_getRetval)
    def wait_for(self, timeout=None):
        if timeout is None or timeout < 0:
            # Use the parent call.
            try:
                out, err = self.communicate()
                self.__finalize()
                return self.returncode, out, err
            except OSError as ex:
                # If the process has already ended, that is fine. This is
                # possible when wait is called from a different thread.
                if ex.errno != 10:  # No child process
                    raise
                return self.returncode, "", ""
        try:
            out, err = self.communicate(timeout=timeout)
            self.__finalize()
            return self.returncode, out, err
        except subprocess.TimeoutExpired:
            self.__finalize()
            raise ProcessError(
                "Process timeout: waited %d seconds, "
                "process not yet finished." % (timeout)
            )
    def kill(self, exitCode=-1, sig=None):
        if sig is None:
            sig = signal.SIGKILL
        try:
            if self.__use_killpg:
                os.killpg(self.pid, sig)
            else:
                os.kill(self.pid, sig)
        except OSError as ex:
            self.__finalize()
            if ex.errno != 3:
                # Ignore:   OSError: [Errno 3] No such process
                raise
        self.returncode = exitCode
        self.__finalize()
    def commandline(self):
        """returns string of command line"""
        if isinstance(self.__cmd, six.string):
            return self.__cmd
        return subprocess.list2cmdline(self.__cmd)
    __str__ = commandline
def execute_and_out(command, timeout=None):
    print("Executing command: {}".format(command))
    process = ProcessOpen(command)
    try:
        #logger.trace("Timeout = {}".format(timeout))
        ret, out, err = process.wait_for(timeout=timeout)
        return ret, out, err
    except ProcessError:
        print("{} command timeout".format(command))
        process.kill()
        return errno.ETIMEDOUT, "", ""
def execute(command, timeout=None):
    ret, _, _ = execute_and_out(command, timeout=timeout)
    return ret
def pre_tasks():
```

```
    pass
def srlinux():
    nos_install()
    nos_configure()
def post_tasks():
    pass
def nos_install():
    cmd = 'ztp image upgrade --imageurl {} --md5url {}'.format(srlinux_image_url, srlinux_
image_md5_url)
    ret,out,err = execute_and_out(cmd)
def nos_configure():
    cmd = 'ztp configure-nos --configurl {}'.format(srlinux_config_url)
    ret,out,err = execute_and_out(cmd)
def main():
    pre_tasks()
    srlinux()
    post_tasks()
if __name__ == '__main__':
    main()
```

Example: Python provisioning script (in-band ZTP)

```
import ztpclient

def nos_install(client):
    image_url = "http://192.168.100.25/images/srlinux-24.10.1-491.bin"
    md5_url = "http://192.168.100.25/images/srlinux-24.10.1-491.bin.md5"
    upgrade = client.image_upgrade(image_url, md5_url)
    if upgrade:
        return int(upgrade['status'])
    return -1

def nos_configure(client):
    srlinux_config_url = "https://192.168.100.25/srlinux/config.json"
    config = client.configure(srlinux_config_url)
    if config:
        return int(config['status'])
    return -1

def is_autoboot_enabled(client):
    return client.option_list()['message']['autoboot']


def main():
    client = ztpclient.APIClient()
    if is_autoboot_enabled(client):
        if nos_install(client):
            raise Exception("nos install failed")
        if nos_configure(client):
            raise Exception("nos configure failed")
    else:
        # Post upgrade steps
        pass


if __name__ == '__main__':
    main()
```

6.4.3 Configuring the ZTP timeout value using the provisioning script

The ZTP process sends DHCP discovery messages on all ports within a ZTP cycle. Every time the DHCP discovery timeout expires and a DHCP offer has not been received, the DHCP discovery process reinitiates on the port until the ZTP timeout expires.

The timeout value can be set using the:

- ZTP CLI (see procedure Configuring options in the grub.cfg using ZTP CLI)
- SR Linux CLI (see procedure Configuring options in the grub.cfg using SR Linux CLI)

6.4.4 Configuring options in the grub.cfg using ZTP CLI

Several options can be manually configured in the grub.cfg using the ZTP CLI at the console. The command has the following format:

```
# ztp option <command> [<arguments>]
```

where *command* must be one of the following:

Table 9: Configuring options

Command	Description
autoboot	Enables or disables the auto-boot flag
bootintf	Specifies boot interface options
clientid	Sets the client ID to a chassis MAC address or serial ID
downgrade	Indicates whether NOS downgrade is allowed
duration	Specifies the ZTP timeout value and number of retry attempts
formatovl	Indicates the format overlay file system on the next reboot
formatsrletc	Indicates the format `/etc/opt/srlinux` overlay file system
formatsrlopt	Indicates the format `/opt/srlinux overlay` file system
list	Displays the current value for each of the command options
nosinstall	Specifies whether a NOS upgrade should be performed as part of the ZTP process
reload	Reloads the config and updates the grub from the config
srlflags	Sets the debug flag in cmdline to a specified value
ztpmode	Specifies the ZTP mode options

Table 10: ZTP CLI: option command examples describes examples of **ztp option** commands and available arguments.

Table 10: ZTP CLI: option command examples

Command and description	Command syntax
autoboot Enable or disable the auto-boot flag	`ztp option autoboot --status [enable | disable]`
bootintf Specify the boot interface to send DHCP over	`ztp option bootintf --intf <name of interface>`
bootintf Remove the previously set boot interface	`ztp option bootintf --remove`
duration Set the ZTP timeout value	`ztp option duration --timeout <integer in seconds>` Default = 3600, range = 200 to 3600
duration Set the number of ZTP retry attempts	`ztp option duration --retry <integer>` Default = 3, range = 1 to 10
clientid Set the client ID to a serialID	`ztp option clientid --type serialid`
nosinstall Enable or disable NOS upgrade flag	`ztp option nosinstall --status [enable | disable]`
list Display the current value of each option	`ztp option list`
ztpmode Specify the ZTP mode option	`ztp option ztpmode --value [ztp | inband | ooband | secure | localdisk]` Default = *ztp*
ztpmode Remove the previously set ZTP mode and revert to default (*ztp*).	`ztp option ztpmode --delete`

Example output

The following is an example output of the **ztp option list** command:

```
# ztp option list
+--------------+----------+
| Name         | Value    |
+--------------+----------+
| autoboot     | False    |
| nosinstall   | False    |
```

```
| bootintf     | mgmt0    |
| timeout      | 3600     |
| retry        | 3        |
| clientid     | serialid |
| downgrade    | True     |
| formatovl    | False    |
| formatsrlopt | True     |
| formatsrletc | False    |
| srlflags     | None     |
| ztpmode      | ztp      |
+--------------+----------+
```

6.4.5 Managing images using ZTP CLI

The ZTP CLI **ztp image** command can be used to activate, delete, list, or perform an upgrade at the console. The following command format is used:

`# ztp image <command> [<arguments>]`

where *command* must be one of the following:

Table 11: Image command descriptions

Command	Description
activate	Activate a specific image
bootorder	Configure a grub entry to match the boot order as passed
delete	Delete a specific image
list	List all available NOS images
upgrade	Perform an upgrade based on provided parameters
version	Extract the version from a specific filename

Table 12: ZTP CLI: ztp image command examples describes examples of **ztp image** commands and available arguments.

Table 12: ZTP CLI: ztp image command examples

Command and description	Command syntax
activate Activate a specific NOS image	`ztp image activate --version <build version> [--no-reboot]` **Note:** `--no-reboot` means do not reboot after activate to take new build into use.
bootorder Configure the bootorder	`ztp image bootorder --version <build version 1> --version <build version 2> --version <build version 3>`

Command and description	Command syntax
	Note: `--version` means the image version order that booting is attempted, which allows up to 3 versions.
delete Delete a specific NOS image	`ztp image delete --version <build version>` **WARNING:** Active image must not be deleted.
list List all available NOS images	`ztp image list`
upgrade Download an image for NOS upgrade	`ztp image upgrade --imageurl <URL to download image>`
upgrade Download an md5 file for NOS upgrade	`ztp image upgrade --md5url <URL to download md5 file>`
upgrade Do not reboot after an upgrade install	`ztp image upgrade --no-reboot`

version
Extract a version
	`ztp image version --filename <filename path> [--format <format type>]`
	Example commands:
	`ztp image version --filename ./config --format json`
	`ztp image version --filename ./srlinux-20.6.1-12617.tar`

Example outputs

Example output (list images):

```
[root@localhost ~]# ztp image list
[ 1638.035200] EXT4-
fs (sdb2): mounted filesystem with ordered data mode. Opts: (null)
+--------------+
| Versions     |
+--------------+
| 20.6.1-10654* |
| 20.6.1-10587  |
| 20.6.1        |
+--------------+
```

Example output (activate image):

```
[root@localhost ~]# ztp image list
```

```
[  227.172007] EXT4-
fs (sdb2): mounted filesystem with ordered data mode. Opts: (null)
+--------------+
| Versions     |
+--------------+
| 20.6.1-10587* |
| 20.6.1-10654  |
| 20.6.1        |
+--------------+
```

Example output (version):

```
[root@localhost ~]# ztp image version --filename ./srlinux-20.6.1-12617.tar
+-------------+---------+
| version     | message |
+-------------+---------+
| 20.6.1-12617 | None   |
+-------------+---------+
```

6.4.6 Configuring the NOS using ZTP CLI

The ZTP CLI **image** command can be used to push the configuration from the console when SR Linux is not operational.

The following is an example showing how to configure the SR Linux with a configuration downloaded with a user-provided URL:

`ztp configure-nos --configurl <URL to download configuration>`

6.4.7 Redownloading the executable files with ZTP CLI

Executable files (scripts and configurations) can be redownloaded if required. The redownload can be performed at the console using the ZTP CLI.

The following is an example showing how to run the provisioning script with a user-provided URL:

`ztp provision --url <URL where files should be downloaded from>`

6.4.8 Starting, stopping, and restarting a ZTP process using ZTP CLI

The ZTP process can be manually started, stopped, and restarted using a ZTP CLI command at the console. The following command format is used:

`# ztp service <command> [<arguments>]`

where *command* must be one of the following:

Table 13: Starting, stopping, and restarting command descriptions

Command	Description
start	Starts ZTP process if not currently running
stop	Stops ZTP process if already running

Command	Description
restart	Stops ZTP process (if running), and then restarts the process

Table 14: ZTP CLI: ztp service command examples describes examples of **ztp service** commands and available arguments.

Table 14: ZTP CLI: ztp service command examples

Command and description	Command syntax
start Start the ZTP process	`ztp service start`
start Start the ZTP process and enable/disable auto-boot	`ztp service start --autoboot [enable | disable]`
stop Stop the ZTP process	`ztp service stop`
stop Stop the ZTP process and disable auto-boot	`ztp service stop --autoboot disable`
restart Restart the ZTP process	`ztp service restart`
restart Restart the ZTP process and enable/ disable auto-boot	`ztp service restart --autoboot [enable | disable]`

6.4.9 Checking the status of a ZTP process using ZTP CLI

The ZTP process status can be manually checked using the ZTP CLI command at the console.

The following is an example showing how to check the status of the ZTP process:

ztp service status

Example:

```
# ztp service status
+---------+----------+
| Service | Status   |
+---------+----------+
| ztp     | Active   |
+---------+----------+
#
```

6.4.10 Configuring options in the grub.cfg using SR Linux CLI

Several auto-boot related options can be manually configured in the grub.cfg using the SR Linux CLI when SR Linux is operational. The command has the following format:

```
# system boot autoboot <command> [<arguments>]
```

where *command* must be one of the following:

Table 15: Configuring options

Command	Description
admin-state	Enables or disables the auto-boot functionality
interface	Sets the interface used for the auto-boot functionality
timeout	Sets the timeout for each auto-boot attempt
attempts	Sets the amount of auto-boot executions to try before rebooting the system
client-id	Sets the client ID to use on outgoing DHCP requests

Table 16: SR Linux CLI: autoboot commands for grub.cfg update examples describes examples of SR Linux **autoboot** commands and available arguments.

Table 16: SR Linux CLI: autoboot commands for grub.cfg update examples

Command and description	Command syntax
admin-state Enable or disable the auto-boot flag	`system boot autoboot admin-state [enable | disable]` Default = enable
interface Specify the boot interface to send DHCP over	`system boot autoboot interface <name of interface>` Default = mgmt0
timeout Set the ZTP timeout value	`system boot autoboot timeout <integer in seconds>` Default = 3600, range = 200 to 3600
attempts Set the number of ZTP retry attempts	`system boot autoboot attempts <integer>` Default = 3, range = 1 to 10

6.4.11 Specifying the image, kernel, or RAM to boot the system using SR Linux CLI

Users can specify an ordered list of local image versions to boot the system using the SR Linux CLI when SR Linux is operational. This directly translates into boot configuration in the grub, where the images are tried in the order specified by the user. The command has the following format:

```
# tools system boot <command> [<arguments>]
```

where *command* must be the following:

Table 17: Specifying the image, kernel, or RAM

Command	Description
image	User-specified ordered list of local image versions used to boot the system

Table 18: SR Linux CLI: image and kernel boot command example describes an example of the SR Linux **image** and **kernel boot** command and available argument.

Table 18: SR Linux CLI: image and kernel boot command example

Command and description	Command syntax
image Specify an ordered list of images to boot the system	`tools system boot image <ordered list of images>` **Note**: Up to 3 versions can be specified.

6.4.12 Starting, stopping, and restarting a ZTP process using SR Linux CLI

The ZTP process can be manually started, stopped, and restarted using the SR Linux CLI when SR Linux is operational. The following command format is used:

```
# tools system boot autoboot <command>
```

where *command* must be one of the following:

Table 19: Commands

Command	Description
execute-script	Executes a specified script as if it were received during auto-boot
start	Starts a ZTP process if not currently running
stop	Stops a ZTP process if already running
restart	Stops an in-progress auto-boot process, then initiates another

Table 20: SR Linux CLI: start, stop, and restart process command examples describes examples of SR Linux **start**, **stop**, and **restart** commands and available arguments.

Table 20: SR Linux CLI: start, stop, and restart process command examples

Command and description	Command syntax
execute-script Execute a specified script	`tools system boot autoboot execute-script <URL to the script>`
start Start the ZTP process	`tools system boot autoboot start`
stop Stop the ZTP process	`tools system boot autoboot stop`
restart Restart the ZTP process	`tools system boot autoboot restart`

6.4.13 Checking the status of a ZTP process using SR Linux CLI

The ZTP process status can be manually checked using the SR Linux CLI when SR Linux is operational.

The following is an example showing how to check the status of the ZTP process:

 # tools system boot autoboot status

6.5 ZTP CLI and SR Linux CLI command structures

This section describes the ZTP CLI command structure and SR Linux CLI command structure.

6.5.1 ZTP CLI command structure

The following ZTP CLI commands are available at the console:

```
ztp
    – chassis
        – control
            – [--format table | json]
        – linecards
            – [--format table | json]
    configure-nos
        – --configurl <URL to download configuration>
    image
        – activate
            – --version <build version>
                – [--no-reboot]
```

```
        – bootorder
            – --version <up to 3 build versions>
        – delete
            – --version <build version>
        – list
            – [--format table | json]
        – upgrade
            – --imageurl <URL to download image> --md5url <URL to download md5 file>
                – [--no-reboot]
                – [--skip-check]
                – [--not-active]
        – version
            – --filename <name>
                – [--format table | json]
    option
        – autoboot
            – --status enable | disable
        – bootintf
            – --intf <boot interface>
        – clientid
            – --type serialid
        – downgrade
            – --status enable | disable
        – duration
            – --timeout <seconds> --retry <integer>
        – formatall
            – --status enable | disable
        – formatovl
            – --status enable | disable
        – formatsrletc
            – --status enable | disable
        – formatsrlopt
            – --status enable | disable
        – grubopt
            – [--key <text>]
            – [--value <text>]
            – [--delete]
        – list
            – [--format table | json]
        – nosinstall
            – --status enable | disable
        – reload
        – srlflags
            – [--value <text>]
            – [--delete]
    provision
        – --url <URL to download provisioning script>
    service
        – restart
            – --autoboot enable | disable
        – start
            – --autoboot enable | disable
        – status
            – [--format table | json]
        – stop
            – --autoboot enable | disable
```

6.5.2 SR Linux CLI command structure

The following SR Linux auto-boot related tool commands are available when the SR Linux is operational:

```
system
   – boot
        – autoboot
            – admin-state
            – attempts
            – client-id
            – interface
            – status
            – timeout
        – image
```

```
tools
    – system
        – boot
            – autoboot
                – execute-script
                – restart
                – start
                – status
                – stop
```

Refer to the *SR Linux Data Model Reference Guide* for additional information about SR Linux commands and parameter descriptions.

6.6 DHCP option based ZTP selection

During ZTP, devices with factory-provisioned Initial Device Identity (IDevID) include a **secure** option in DHCP requests, even if the boot parameter (**ztpmode**) in `grub.cfg` file is set to the default value (**ztp**). For information about configuring ZTP bootmode options using the ZTP CLI, see Configuring options in the grub.cfg using ZTP CLI.

The device acts according to the DHCP server response:

- If the DHCP server returns a redirect option containing a URL, the URL schema determines the next step. If the URL starts with `bootz://`, the device launches the BootZ process.
- If the DHCP response includes only standard options (Options 66 and 67, Option 67 alone, or Option 43), normal ZTP is performed.
- If the DHCP response includes both options, the redirect option takes precedence and the device launches the BootZ process.

Note: If the device does not have a TPM, or the IDevID cannot be read from the TPM, it does not include the secure option in its DHCP request, and the normal ZTP is performed.

On chassis devices, ZTP performs out-of-band provisioning with secure options in the following order:

- The device first attempts out-of-band IPv4 provisioning using the chassis serial number.
- If unsuccessful, it uses the card serial number for out-of-band IPv4 provisioning.
- Next, the device attempts out-of-band IPv6 provisioning using the chassis serial number.

- Finally, it uses the card serial number for out-of-band IPv6 provisioning.

This process repeats until at least one ZTP attempt succeeds.

Supported platform

The DHCP option-based ZTP selection is supported on the following platforms:

- 7250 IXR-6 (Gen 2c+)
- 7250 IXR-10 (Gen 2c+)
- 7250 IXR-6e (Gen 3)
- 7250 IXR-10e (Gen 3)
- 7250 IXR-18e
- 7250 IXR-X1b
- 7250 IXR-X3b

6.7 ZTP CLI and SR Linux CLI command structures

This section describes the ZTP CLI command structure and SR Linux CLI command structure.

6.7.1 ZTP CLI command structure

The following ZTP CLI commands are available at the console:

```
ztp
    – chassis
        – control
            – [--format table | json]
        – linecards
            – [--format table | json]
    configure-nos
        – --configurl <URL to download configuration>
    image
        – activate
            – --version <build version>
                – [--no-reboot]
        – bootorder
            – --version <up to 3 build versions>
        – delete
            – --version <build version>
        – list
            – [--format table | json]
        – upgrade
            – --imageurl <URL to download image> --md5url <URL to download md5 file>
                – [--no-reboot]
                – [--skip-check]
                – [--not-active]
        – version
            – --filename <name>
                – [--format table | json]
    option
        – autoboot
            – --status enable | disable
        – bootintf
```

```
            – --intf <boot interface>
        – clientid
            – --type serialid
        – downgrade
            – --status enable | disable
        – duration
            – --timeout <seconds> --retry <integer>
        – formatall
            – --status enable | disable
        – formatovl
            – --status enable | disable
        – formatsrletc
            – --status enable | disable
        – formatsrlopt
            – --status enable | disable
        – grubopt
            – [--key <text>]
            – [--value <text>]
            – [--delete]
        – list
            – [--format table | json]
        – nosinstall
            – --status enable | disable
        – reload
        – srlflags
            – [--value <text>]
            – [--delete]
    provision
        – --url <URL to download provisioning script>
    service
        – restart
            – --autoboot enable | disable
        – start
            – --autoboot enable | disable
        – status
            – [--format table | json]
        – stop
            – --autoboot enable | disable
```

6.7.2 SR Linux CLI command structure

The following SR Linux auto-boot related tool commands are available when the SR Linux is operational:

```
system
   – boot
        – autoboot
            – admin-state
            – attempts
            – client-id
            – interface
            – status
            – timeout
        – image
```

```
tools
    – system
        – boot
            – autoboot
                – execute-script
                – restart
```

```
                - start
                - status
                - stop
```

Refer to the *SR Linux Data Model Reference Guide* for additional information about SR Linux commands and parameter descriptions.

6.8 ZTP using USB/SD in a non-secure environment

When deploying ZTP across an insecure network, the technician prepares a removable storage device, such as a USB drive or SD card, containing all required bootstrapping data for the node. This includes the NOS image, bootstrap configuration, optional pre-config and post-config scripts, and a ZTP manifest (`nokia_ztp_config.yml`) that tags each file and defines the execution order.

After the technician mounts the nodes in the rack and powers them on, the ZTP client checks for a removable storage device.

- If a removable storage device is found, ZTP scans all partitions for the file `nokia_ztp_config.yml` at each root. If the file is found, ZTP starts the local disk-based bootstrapping.
- If no removable storage device is found, the client falls back to the standard DHCP-based ZTP process (`oobv4`, `oobv4cardserial`, `oobv6`, `oobv6cardserial`).

The following is a sample ZTP manifest file (`nokia_ztp_config.yml`).

```
#ztp-config
config-version: 1
onboarding-information:
  boot-image:  "images/srlinux.bin"
  image-verification:
    hash-algorithm:              "sha256"
    hash-value:
 "59:ab:ed:9b:e6:93:5a:88:cd:49:f1:90:54:99:b9:81:02:6a:eb:62:9a:9d:eb:2a:d2:78:c5:19:64:c5:b5:78"
  pre-configuration:
    - "scripts/pre_config.sh"
    - "scripts/pre_config.py"
  configuration:                 "cfg/config.json"
  post-configuration:     "scripts/post_config.sh"
```

- **ztp-config** : mandatory preamble that identifies the file as a ZTP configuration and has bootstrap information.
- **config-version**: must be set to 1; this is a mandatory parameter.
- The following parameters are optional:
  - **hash-algorithm**: default is SHA-256; MD5 is also supported.
  - **pre-configuration**: refers to the list of scripts executed after an upgrade-reboot and before SR Linux starts.
  - **post-configuration**: refers to the list of scripts executed after SR Linux has started, and the configuration is applied.
  - **configuration**: specifies the location of the SR Linux configuration relative to the root partition and the manifest file (`nokia_ztp_config.yml`).

- **boot-image**: specifies the location of the SR Linux image relative to the root partition and the manifest file (`nokia_ztp_config.yml`).

After parsing the manifest file (`nokia_ztp_config.yml`), the client uses the existing ZTP framework with the Python provisioning script orchestrating the bootstrapping sequence as in a standard DHCP-based ZTP. For more information, see Python provisioning script.

7 BootZ

Note:
Ensure you obtain a bootstrapping RTU (Right to Use) license, and purchase new devices with an OS kit that has BootZ enabled by default instead of ZTP. For information about supported platforms, see Supported Platforms.

BootZ is a gRPC-based protocol used to bootstrap a network device securely.

To enable device bootstrapping, BootZ defines the boot process and provides a specification that enumerates the data elements that allows a vendor-agnostic implementation.

The SR Linux implementation of BootZ is currently a partial application of the BootZ specification and is evolving to meet all aspects of the BootZ protocol .

The SR Linux BootZ implementation uses DHCP and bootstrap servers to initiate the bootstrapping of source data. Currently, only the bootstrap process from DHCP discovery to successful processing of the bootstrap data is supported.

In BootZ implementation, the DHCP server provides the CPM with unsigned redirect information, which includes a list of bootstrap servers. The CPM then uses this information to execute the bootstrapping RPCs against the list of bootstrap servers.

Supported Platforms

BootZ is supported on these platforms:

- 7250 IXR-6e CPM4 with Root of Trust
- 7250 IXR-10e CPM4 with Root of Trust
- 7250 IXR-18e

7.1 BootZ components

On-boarding device

Nokia router that you want to provision and connect to your network.

DHCP server

The DHCP server provides the node with the location of the bootstrap BootZ server.

BootZ bootstrap server

The BootZ bootstrap server hosts the bootstrapping data, which includes the OS version and initial device configuration. The bootstrap servers use gRPC methods to communicate with the on-boarding devices.

The on-boarding device uses RPC `GetBootstrappingData` to initiate the BootZ process with the BootZ server. The RPC method uses the structure, `GetBootstrapDataRequest` for the request data and the structure, `GetBootstrapDataResponse` for the response data. The structure, `GetBootstrapData Response` contains the bootstrapping data. It also includes the initial gNSI artifacts, such as, certz, pathz,

authz, and credential artifacts, which are required to enable the device to proceed with enrollment and attestation or to bring the device fully into a production state.

The on-boarding device uses the RPC `ReportProgress` to report its bootstrapping progress to the BootZ server.

Bootstrapping artifacts

Bootstrapping artifacts are as follows:

- Redirect information: Refers to the data provided to an on-boarding device that directs it to an alternative server or endpoint for obtaining further configurations or bootstrapping data.
- On-boarding information: On-boarding information supplies the data required for a device to bootstrap and establish secure connections with other systems.
- Ownership Voucher: To support BootZ, an Ownership Voucher (OV) must be obtained by submitting a request to Nokia Support. The request must include the Pinned Domain Certificate (PDC) and the order details containing the router serial number provided to Nokia during the OV request. Nokia generates an ownership voucher and sends it back as a response to the ownership voucher request.
- Ownership certificate: Represents an X.509 certificate that binds an owner identity to a public key, allowing a device to validate a signature on the on-boarding information artifact. It is provided to the device via the bootstrap server. The ownership certificate is validated against the PDC, which is contained in the Ownership Voucher.

7.2 BootZ process

Prerequisites

1. Ensure that you obtain a bootstrapping RTU (Right to Use) license and purchase new devices with an OS kit, with BootZ enabled by default, instead of ZTP.
2. Ensure that you obtain an Ownership Voucher (OV) by submitting a request to Nokia Support. The request must include the Pinned Domain Certificate (PDC), along with the order details containing the serial numbers of the CPM.
3. Ensure that DHCP and BootZ bootstrap servers are configured in the network.

Procedure

Step 1. At the installation site, the BootZ OS kit triggers the initialization of the secure bootstrapping process.

Step 2. DHCP discovery of BootZ Bootstrap server

- **a.** The node boots up and initiates a DHCP request to the DHCP server.
- **b.** The DHCP server assigns an address to the requesting node and provides a list of bootstrap server URIs. The DHCP response contains the option code, `OPTION_V4_SZTP_REDIRECT(143)` or `OPTION_V6_SZTP_REDIRECT(136)`. The response code, `OPTION_V4_SZTP_REDIRECT(143)` is the DHCP v4 code for IPv4 addressing, and `OPTION_V6_SZTP_REDIRECT(136)` is the DHCP v6 code for IPv6 addressing. The URI is in the following format: `bootz://<host or ip>:<port>`.

Step 3. Bootstrapping service

a. The router sends the `GetBootstrappingData` RPC to the bootstrap server obtained from the DHCP server. The device establishes a gRPC connection to the BootZ server. The device must use its unique IDevID certificate to establish and secure the TLS connection to the BootZ server.The IDevID certificate is permanently programmed in the TPM of the control card. During the BootZ process, the IDevID certificate is fetched from the TPM. For more information on TPM, see TPM Keys (IDevID and IAK).

Note: In platforms with dual IDevIDs (for example, 7250 IXR-18e), the BootZ process initially attempts to connect to the gRPC server using the primary IDevID. If the connection fails, the device automatically retries using the secondary IDevID. No additional configuration or manual intervention is required from the operator or user to switch between the IDevIDs to complete the BootZ process.

b. The BootZ server sends a response in the structure `GetBootstrapDataResponse`. The GetBootstrapDataResponse contains the bootstrapping data and includes the initial gNSI artifacts, such as certz, pathz, authz, credential artifacts, and initial device configurations. The bootstrap data is signed by an ownership certificate. After the on-boarding data is validated, the device processes the on-boarding data, which includes the OS version, link/URL to download the OS image, and initial device configuration scripts.

c. The device validates the Ownership Voucher using one of the stored trust anchor certificates and then uses the Pinned Domain Certificate in the ownership voucher to authenticate the ownership certificate. Before accepting on-boarding data, the device verifies if the on-boarding data is signed by the ownership certificate, which was previously validated by the device. If the signature verification fails, the Bootstrap process restarts from step 2.

Step 4. Report progress

a. When the router obtains the on-boarding information, it reports the bootstrapping progress to the BootZ server using RPC `ReportProgress`.

Step 5. The router downloads the OS image, verifies its hash, checks if the image version from bootstrap data differs from the current running version, and installs the OS if needed (if a new version is different from the currently running version). The router then runs the post-installation scripts and becomes operational. A reboot is required if the image version differs from the current running image. The post-installation script is run after the device boots with the new version.

Step 6. After the device boots up with the new image, it runs the post-installation script provided by the bootstrap server and sends a final report.

8 Appendix: ZTP Python library

This appendix describes the importable ZTP Python library.

For more information about the ZTP process, see Zero Touch Provisioning.

8.1 ZTPClient

The ZTPClient communicates with the SR Linux ZTP process. The APIClient is the core object of the ZTPClient. Each use of the ZTPClient passes through a call to one of its methods.

The path to the API client class is as follows:

class ztpclient.ztpclient.APIClient(*base_url=None*)

Example

```
def __init__(self):
        self.client = ztpclient.APIClient()
    def get_option(self, item):
        ret = self.client.option_list()
        return ret['message'].get(item, None)
    def find_current_version(self):
        response = self.client.image_list()
        if response:
            image = response['message']
            if image and isinstance(image, list) and len(image) > 0:
                return image[0].replace('*', '')
        return None
    def perform_ztp(self):
        self.nos_install()
        self.nos_configure()
        self.disable_autoboot()
    def nos_install(self):
        ret = self.client.image_upgrade(srlinux_image_url, srlinux_image_md5_url)
        if ret:
            return int(ret['status'])
        return -1
    def nos_configure(self):
        ret = self.client.configure(srlinux_config_url)
        if ret:
            return int(ret['status'])
        return -1
    def disable_autoboot(self):
        ret = self.client.option_autoboot(ztpclient.ZtpStatus.disable)
        if ret:
            return int(ret['status'])
        return -1
    if __name__ == '__main__':
        ztp = ZTP()
        ztp.perform_ztp()
```

8.2 Functions

This section describes the possible functions.

8.2.1 chassis_control()

Lists control card information.

Table 21: chassis_control()

Information	Description
Arguments	—
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise. The **message** attribute contains dictionary with control card information.

Example	`>>> client.chassis_control()`
	`{u'status': 0, u'message': {u'operation': u'active'}}`
	`>>> client.chassis_control()`
	`{u'status': 0, u'message': {u'operation': u'standby'}}`

8.2.2 chassis_linecards()

Lists line card information for the chassis.

Table 22: chassis_linecards()

Information	Description
Arguments	—
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise. The **message** attribute contains list of dicts, where list item is dict with line card information.

Example	`>>> client.chassis_linecards()`
	`{u'status': 0, u'message': [{u'card_type': 127, u'card_name': u'imm32-`
	`100g-qsfp28+4-400g-qsfpdd', u'slot_num': 1}, {u'card_type': 0,`
	`u'card_name': u'empty', u'slot_num': 2}, {u'card_type': 0,`
	`u'card_name': u'empty', u'slot_num': 3}, {u'card_type': 127,`
	`u'card_name': u'imm32-100g-qsfp28+4-400g-qsfpdd', u'slot_num': 4}]}`

8.2.3 configure(configurl)

Downloads the configuration from a specific **configurl** and applies the configuration to SR Linux. If SR Linux services are not running, the services are started and the configuration is applied.

Table 23: configure(configurl)

Information	Description
Arguments	**configurl** (string): the URL from where the configuration will be downloaded
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.
Example	—

8.2.4 image_activate(version)

Reboots the chassis to the image version provided. If the current active version is the same as the specified **version**, no action is performed. If there is no image in the chassis of the specified **version**, no action is performed. If the specified **version** is available, the chassis will be rebooted to that **version**.

Table 24: image_activate(version)

Information	Description
Arguments	**version** (string): the image version
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.  **Note:** This API may result in a chassis reboot to activate the image version.

Examples	`>>> client.image_list()`
	`{u'status': 0, u'message': [u'20.6.1-18836*', u'20.6.1-17740',`
	`u'20.6.1-17738']}`
	`>>> client.image_activate('20.6.1-3333')`
	`{u'status': 127, u'message': u'20.6.1-3333 is not available'}`
	`>>> client.image_activate('20.6.1-18836')`
	`{u'status': 127, u'message': u'20.6.1-18836 is current active version.`
	`No additional change required'`

8.2.5 image_bootorder(bootorder)

Sets the image bootorder in the Grub configuration. On the next reboot, the chassis reboots to the first image in the list.

Table 25: image_bootorder(bootorder)

Information	Description
Arguments	**bootorder** (list): the image version list
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.

Examples	`>>> client.image_bootorder(['20.6.1-18836','20.6.1-17740','20.6.1-17738'])`
	`{u'status': 0, u'message': None}`
	`>>> client.image_bootorder('20.6.1-18836,20.6.1-17740,20.6.1-17738')`
	`{u'status': 0, u'message': None}`

8.2.6 image_delete(version)

Removes the specified image **version** from the chassis. If the specified **version** is not available in the chassis, no action is performed. If the specified **version** is the current active version in the chassis, no action is performed.

Table 26: image_delete(version)

Information	Description
Arguments	**version** (string): the image version
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.

Examples	`>>> client.image_list()`
	`{u'status': 0, u'message': [u'20.6.1-18836*', u'20.6.1-17740', u'20.6.1-17738']}`
	`>>> client.image_delete('20.6.1-3333')`
	`{u'status': 0, u'message': u'20.6.1-3333 version not available'}`
	`>>> client.image_delete('20.6.1-18836')`
	`{u'status': 127, u'message': u'Cannot remove active version'}`

8.2.7 image_list()

Lists all currently available image versions on the hardware.

Table 27: image_list()

Information	Description
Arguments	—

Information	Description
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise. The **message** attribute contains the list of images. The list item followed by an asterisk (*) indicates the current active image version. **Note:** The **image_list** does not indicate the boot order.

Examples	`>>> client.image_list()`
	`{u'status': 0, u'message': [u'20.6.1-18836*', u'20.6.1-17740',`
	`u'20.6.1-17738']}`

8.2.8 image_upgrade(image_url, md5_url, options)

Performs an image upgrade.

Table 28: image_upgrade(image_url, md5_url, options)

Information	Description
Arguments	**image_url** (string): the URL from where the image should be downloaded **md5_url** (string): The URL from where the pre calculated md5sum of the image should be downloaded. After the image is downloaded, the calculated md5sum is checked against the downloaded md5sum. If the values do not match, the image upgrade is discarded. **no_reboot** (boolean): If set to true, a chassis reboot is not triggered after an image upgrade. The new image will not be taken into use until the next reboot. The default is false. **skip_check** (boolean): If set to true, the status check of the **autoboot** parameter is skipped, and a forced upgrade is performed. If set to false, the image upgrade will only be performed if **autoboot** is enabled. The default is false. **not_active** (boolean): If set to true, after an image install, the image will not be marked as the active image (that is, will not reboot to the upgrade image). The current working image is still marked as active. The default is false. **Note:** Based on the setting and outcome, the chassis can be rebooted when invoking this API. **Note:** To perform ZTP, the **autoboot** flag must be enabled.
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.
Examples	—

8.2.9 option_autoboot(status)

Sets the **autoboot** option status. This option determines if **autoboot** should be performed during ZTP. If disabled, ZTP skips all steps and starts the SR Linux application.

Table 29: option_autoboot(status)

Information	Description
Arguments	**status** (ZtpStatus): **ztpclient.ZtpStatus.enable** to enable the option, **ztpclient.ZtpStatus.disable** otherwise
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.
Examples	`>>> client.option_autoboot(ztpclient.ZtpStatus.enable)` `{u'status': 0, u'message': None}`

8.2.10 option_bootintf(interface)

Sets the interface to be used by ZTP in various procedures. The default value is **mgmt0**.

Table 30: option_bootintf(interface)

Information	Description
Arguments	**interface** (string): the Linux network interface name
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.
Examples	`>>> client.option_bootintf('mgmt0')` `{u'status': 0, u'message': None}`

8.2.11 option_clientid(type)

Sets the client ID used by ZTP when performing a DHCP request. The possible values are **serialid** and **mac**. When **serialid** is selected, the chassis serial number is used as the client ID in the DHCP request. When **mac** is selected, the Linux interface hardware address (chassis MAC address) is used as client identifier.

Table 31: option_clientid(type)

Information	Description
Arguments	**type** (ZtpClientId): the client identifier type

Information	Description
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.

Examples	`>>> client.option_clientid(ztpclient.ZtpClientId.serialid)`
	`{u'status': 0, u'message': None}`
	`>>> client.option_clientid(ztpclient.ZtpClientId.mac)`
	`{u'status': 0, u'message': None}`

8.2.12 option_downgrade(status)

Sets the **downgrade** option status of ZTP. When enabled, the option allows ZTP to perform a downgrade of the image (that is, move from a later version image to an earlier version). When the option is disabled, only upgrades are allowed.

Table 32: option_downgrade(status)

Information	Description
Arguments	**status** (ZtpStatus): **ztpclient.ZtpStatus.enable** to enable the option, **ztpclient.Ztp Status.disable** otherwise
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.
Examples	`>>> client.option_downgrade(ztpclient.ZtpStatus.enable)` `{u'status': 0, u'message': None}`

8.2.13 option_duration(timeout, retry)

Sets the **timeout** and **retry** parameters of the ZTP process. If not successful, the ZTP process keeps retrying for the specified **timeout** seconds. When the timeout is reached, the process stops. If the number of attempts are equal to the **retry** value, the specified action is taken. The default action is to reboot.

Table 33: option_duration(timeout, retry)

Information	Description
Arguments	**timeout** (int): the number of seconds to perform ZTP before it is marked as failed **retry** (int): the number of attempts before stopping the ZTP process
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.
Examples	`>>> client.option_duration(3600,3)`

Information	Description
	`{u'status': 0, u'message': None}`

8.2.14 option_formatovl(status)

Sets the **formatovl** option status of ZTP. When enabled, the option sets the **srl.formatovl** flag in the Grub configuration. On the next reboot, if the **srl.formatovl** flag is set, the NOKIA-DATA overlay file system is formatted. Any change performed on the overlay file system is removed.

Table 34: option_formatovl(status)

Information	Description
Arguments	**status** (ZtpStatus): **ztpclient.ZtpStatus.enable** to enable the option, **ztpclient.Ztp Status.disable** otherwise
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.
Examples	`>>> client.option_formatovl(ztpclient.ZtpStatus.enable)` `{u'status': 0, u'message': None}`

8.2.15 option_formatsrletc(status)

Sets the **formatsrletc** option status of ZTP. When enabled, the option sets the **srl.formatetc** flag in the Grub configuration. On the next reboot, if the **srl.formatetc** flag is set, the NOKIA-ETC overlay file system is formatted. Any change performed on the overlay file system will be removed.

Table 35: option_formatsrletc(status)

Information	Description
Arguments	**status** (ZtpStatus): **ztpclient.ZtpStatus.enable** to enable the option, **ztpclient.Ztp Status.disable** otherwise
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.
Examples	`>>> client.option_formatsrletc(ztpclient.ZtpStatus.enable)` `{u'status': 0, u'message': None}`

8.2.16 option_formatsrlopt(status)

When enabled, the option sets the **srl.formatopt** flag in the Grub configuration. On the next reboot, if the **srl.formatopt** flag is set, the NOKIA-OPT overlay file system is formatted. Any change performed on the overlay file system will be removed.

Table 36: option_formatsrlopt(status)

Information	Description
Arguments	**status** (ZtpStatus): **ztpclient.ZtpStatus.enable** to enable the option, **ztpclient.ZtpStatus.disable** otherwise
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.
Examples	`>>> client.option_formatsrlopt(ztpclient.ZtpStatus.enable)` `{u'status': 0, u'message': None}`

8.2.17 option_list()

Lists all the options of the ZTP process.

Table 37: option_list()

Information	Description
Arguments	—
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.

Examples	`>>> client.option_list()`
	`{u'status': 0, u'message': {u'formatsrletc': False, u'retry': 3,`
	`u'bootintf': u'mgmt0', u'clientid': u'serialid', u'autoboot': False,`
	`u'srlflags': u'no-reboot', u'formatovl': False, u'formatsrlopt':`
	`False, u'timeout': 3600, u'downgrade': True, u'nosinstall': False}}`

8.2.18 option_nosinstall(status)

Sets the **nosinstall** option status. This option determines if an image upgrade should be performed during ZTP. Only the image upgrade step is skipped. All other steps of ZTP are still performed.

Table 38: option_nosinstall(status)

Information	Description
Arguments	**status** (ZtpStatus): **ztpclient.ZtpStatus.enable** to enable the option, **ztpclient.Ztp Status.disable** otherwise
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.
Examples	`>>> client.option_nosinstall(ztpclient.ZtpStatus.enable)` `{u'status': 0, u'message': None}`

8.2.19 provision(provisionurl)

Downloads the provision script from a specific **provisionurl** and executes the script. The script could be either **Python** or **Bash**.

Table 39: provision(provisionurl)

Information	Description
Arguments	**provisionurl** (string): the URL from where the provisioning script will be downloaded
Returns	(dict) The API response as a Python dictionary. The **status** attribute is set to 0 if successful, or a non-zero value otherwise. **Note:** If the script returns a non-zero exit code, the **status** attribute in the return dictionary is set to non-zero. It could be possible that the provisioning script has a chassis reboot command and a chassis will reboot while executing this API. **Note:** To perform ZTP, the **autoboot** flag must be enabled.
Examples	`>>> client.provision('http://135.227.248.118/duts/IDNS1833F0766/` `srlinux_ztp.py')`

8.2.20 service_restart()

Restarts the ZTP service.

Table 40: service_restart()

Information	Description
Arguments	—

Information	Description
Returns	(dict) The API response as a Python dictionary, including the service status in the **message** attribute. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.
Examples	`>>> client.service_restart()` `{u'status': 0, u'message': {u'status': u'Service started'}}`

8.2.21 service_start()

Starts the ZTP service (if not already running).

Table 41: service_start()

Information	Description
Arguments	—
Returns	(dict) The API response as a Python dictionary, including the service status in the **message** attribute. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.
Examples	`>>> client.service_start()` `{u'status': 0, u'message': {u'status': u'Service started'}}`

8.2.22 service_status()

Gets the current status of the ZTP service. The ZTP service will be running as a systemd service. It can be checked manually by executing the **systemctl status ztp** command.

Table 42: service_status()

Information	Description
Arguments	—
Returns	(dict) The API response as a Python dictionary, including the service status in the **message** attribute. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.

Examples	`>>> client.service_status()`
	`{u'status': 0, u'message': {u'status': u'Inactive'}}`
	`>>> client.service_status()`
	`{u'status': 0, u'message': {u'status': u'Active'}}`

8.2.23 service_stop()

Stops the ZTP service (if already running).

Table 43: service_stop()

Information	Description
Arguments	—
Returns	(dict) The API response as a Python dictionary, including the service status in the **message** attribute. The **status** attribute is set to 0 if successful, or a non-zero value otherwise.
Examples	`>>> client.service_stop()` `{u'status': 0, u'message': {u'status': u'Service stopped'}}`

9 Appendix: Migrating from CentOS to Debian OS

The migration process from CentOS to Debian OS involves the following high-level changes:

- The filename format for `.deb` packages differs from that of `.rpm` packages. In the `.deb` format, the filename follows the pattern: *<name>_<version>_<arch>*`.deb`

  Note: Underscores must not be included in *name*, *version*, or *arch* components. The `.rpm` format is different from this, as it uses hyphens to separate components and allows underscores in names.

  In addition to change in package name, this also involves change in package formatting, including a change in the package manager.
- Debian uses the **apt** or **apt-get** command for package management, while CentOS uses **yum** or **dnf**.
- The loading of shell profile files, such as `/etc/profile.d/sr_app_env.sh`, differs between Debian and CentOS. In most cases, the loading process is centrally aligned. However, there can be instances where the environment of an application or script does not meet the expected conditions. For example, the **PATH** variable may not include the **srlinux** paths. In such cases, you can replace **sh** to **bash**, or append **-l** to the shell command, or combine both options, to make it a login shell.

Debian 13 uses Python 3.13.5 by default. The Python upgrade from the previously used 3.6 version to 3.13.5 does not affect the functionality of the following items, unless otherwise specified.

- Event Handler scripts
- NDK

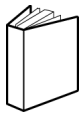
  Note: The new path of the generated NDK protos on Debian is `/usr/lib/python3.13/dist-packages/sdk_protos`. A symlink has been added to refer to the new path from the old one. In future releases, the symlink will no longer be available. Therefore, NDK developers who use on-box generated protos must update their import paths accordingly.

- Python provisioning scripts
- Show commands and CLI plug-ins
- ZTP Python library

However, in the case of custom package development, following must be considered:

- Python packages are installed as `wheels` instead of `eggs`. For instructions about packaging in `wheels` format, see the Wheels binary package documentation.
- The parametrized package replaces the Python module nose-parameterized package. For migration instructions, see the nose-parametrized documentation.

Customer document and product support



Customer documentation

Customer documentation welcome page

Technical support

Product support portal

Documentation feedback

Customer documentation feedback